Available *online* at : http://jnte.ft.unand.ac.id/

Jurnal Nasional Teknik Elektro

| ISSN (Print) 2302-2949 | ISSN (Online) 2407-7267 |

MQTT Broker Optimization: Comparative Analysis of Round Robin and Least Response Time

Samsul Arifin [1], Alfian Wahyu Nugraha [2], Fransiska Sisilia Mukti [2], Sunu Jatmika [1]

[1] *Department of Computer Engineering, Institut Teknologi dan Bisnis Asia Malang, 65113, Indonesia*
[2] *Department of Informatics Engineering, Institut Teknologi dan Bisnis Asia Malang, 65113, Indonesia*

ARTICLE INFORMATION

Received: October 24, 2024
Revised: October 31, 2024
Accepted: November 25, 2024
Available online: November 30, 2024

KEYWORDS

MQTT, Round Robin, Least Response Time, Load Balancing, IoT

CORRESPONDENCE

Phone: +62 83848595545
E-mail: ms.frans@asia.ac.id

A B S T R A C T

Optimizing MQTT broker performance is crucial for maintaining efficient message routing in IoT systems, especially under varying workloads and QoS levels. This study compares the Round Robin (RR) and Least Response Time (LRT) algorithms to evaluate their performance across QoS levels 0, 1, and 2 and client loads ranging from 500 to 2,500 clients. Using Apache JMeter, key metrics such as CPU usage, throughput, delay, jitter, and response time were assessed. LRT was found to excel in enhancing response time and reducing delay, particularly under high client loads and in applications requiring minimal latency. However, this comes at the cost of higher CPU usage under heavy loads. In contrast, RR demonstrated optimal performance in maintaining balanced CPU utilization and predictable performance, though with slightly higher response times. Both algorithms demonstrated linear scalability in throughput, confirming their ability to handle increasing workloads without bottlenecks. These findings offer practical guidance for IoT developers: in latency-sensitive environments such as industrial automation, LRT is preferable due to its low-latency benefits, while RR is better suited for resource-constrained IoT systems like environmental monitoring, where stability and even load distribution are prioritized. The trade-offs identified provide valuable insights for selecting appropriate algorithms based on specific application requirements.

INTRODUCTION

The Internet of Things (IoT) has emerged as a transformative technological paradigm, enabling interconnected smart devices to communicate seamlessly and perform automated tasks across diverse environments. From smart homes and industrial automation to healthcare [1], agriculture [2],[3],[4], and transportation systems, IoT facilitates real-time data exchange, monitoring, and control, significantly enhancing efficiency and decision-making processes [5], [6]. However, the rapid expansion of IoT networks has introduced new challenges, especially in managing large-scale communication between thousands of devices. Efficient communication protocols are vital to ensure smooth data transmission and prevent bottlenecks within such complex networks [7].

Among the available communication protocols, Message Queuing Telemetry Transport (MQTT) has become a popular choice for IoT deployments due to its lightweight messaging architecture. MQTT is designed to operate with minimal overhead, making it suitable for resource-constrained devices and environments with limited bandwidth [8], [9]. Unlike traditional point-to-point communication models, MQTT adopts an asynchronous Publish-Subscribe model, where publishers send messages to topics managed by a broker, and subscribers receive only the messages relevant to their subscribed topics. This model ensures scalable, efficient, and non-blocking communication, which is essential for IoT applications requiring low latency and high throughput [10].

The broker is at the heart of the MQTT protocol, a key component responsible for receiving, storing, and distributing messages between connected devices. The broker is an intermediary, ensuring data flows reliably between publishers and subscribers [11]. However, broker performance is critical, as any degradation—such as overload or latency—can disrupt the entire IoT system, leading to message loss, increased latency, or system unavailability [12]. In large-scale deployments with thousands of devices, the broker may become a single point of failure if it cannot handle the volume of client requests effectively [13], [14].

To address these challenges, load balancing is pivotal in distributing traffic evenly across multiple brokers or servers, ensuring optimal computational resource utilization and system stability[15]. Without an effective load-balancing strategy, brokers risk becoming overloaded, leading to performance bottlenecks, service disruptions, and client disconnections. Load balancing ensures not only that each broker handles an appropriate share of the workload but also that redundancy and fault tolerance are achieved [16]. This mechanism is especially

https://doi.org/10.25077/jnte.v13n3.1260.2024

important for mission-critical IoT systems, where communication delays or failures can have significant consequences [17].

Two prominent load balancing algorithms for MQTT are Round Robin (RR) and Least Response Time (LRT). RR assigns requests sequentially to brokers, promoting a simple and even workload distribution. While easy to implement, RR's static nature limits its adaptability to varying traffic loads, as it does not consider the current capacity or response time of brokers [18]. On the other hand, LRT dynamically routes requests to the broker with the lowest average response time and the fewest active connections, ensuring traffic flows to the most responsive server. This makes LRT ideal for environments with fluctuating workloads, though it introduces more computational overhead to monitor broker performance continuously [19].

Although several studies have explored load balancing in MQTT environments, the focus has primarily been on traditional algorithms like Round Robin, with limited attention to adaptive strategies such as Least Response Time. For instance, Charlie et al. [20] evaluated Round Robin’s performance but did not examine how it affects Quality of Service (QoS), a critical metric for assessing network performance. Similarly, Sepriano et al. [21]compared Round Robin and Least Connection but did not conduct a detailed analysis across multiple QoS levels. Other research, such as Li [22] and Naufal Romiz et al. [23], examined load balancing across protocols like CoAP but lacked focus on MQTT-specific scenarios and adaptive algorithms.

Furthermore, studies such as Mishra et al. [24] have evaluated the performance of various MQTT brokers, including Mosquitto, using metrics like CPU usage, message rate, and latency. While these studies provide valuable insights into broker performance, they do not address the potential benefits of adaptive algorithms like Least Response Time under varying network conditions and client loads. This gap highlights the need for a comprehensive comparison between Round Robin and Least Response Time algorithms better to understand their strengths and weaknesses in MQTT broker optimization.

This research compares the round-robin and Least Response Time (LRT) algorithms to optimize MQTT brokers. It focuses on their performance across different Quality of Service (QoS) levels and client loads. The study addresses a significant gap by exploring adaptive load-balancing strategies for MQTT systems, offering practical insights for designing efficient IoT applications.

One key contribution is the in-depth examination of the LRT algorithm within MQTT brokers, particularly its impact on scalability and responsiveness under high client loads [25]. The results demonstrate that LRT offers improved resource utilization and responsiveness compared to Round Robin, especially at higher client volumes. This exploration highlights the algorithm's potential to enhance the performance of MQTT systems across QoS levels 0, 1, and 2.

Additionally, the research benchmarks both algorithms under various scenarios using Apache JMeter, evaluating performance metrics such as throughput, CPU usage, jitter, delay, and response time. The findings reveal that Round Robin ensures even CPU distribution but may struggle with higher response times under heavy loads. At the same time, despite achieving lower latency and jitter, LRT incurs higher CPU consumption.

The study offers actionable recommendations for IoT developers based on these trade-offs. LRT is ideal for applications requiring low latency and high responsiveness, such as industrial automation and real-time monitoring. In contrast, Round Robin is recommended for resource-constrained environments, prioritizing CPU efficiency and balanced load distribution. The impact of QoS levels on each algorithm’s performance is also discussed, guiding developers in choosing appropriate load-balancing strategies according to specific system needs.

The structure of this paper is organized as follows. Methods provide a detailed description of the research methodology, including the experimental system, algorithms employed, and evaluation methods. Results and Discussion offers a detailed presentation of the experimental outcomes, accompanied by critical analysis and discussion. Finally, the Conclusions summarize the main findings, provide recommendations, and identify potential avenues for future research.

METHODS

This study adopts an experimental design to evaluate the performance of two load-balancing algorithms—RR and LRT—within an MQTT-based communication system. The goal is to determine which algorithm provides better workload distribution across multiple brokers while ensuring optimal Quality of Service (QoS). The study involves system design, implementation, testing, and data analysis to provide insights into the suitability of each algorithm for real-world deployments.

System Architecture and Communication Flow

The system architecture consists of key components: Apache JMeter clients, an NGINX load balancer, and multiple Mosquitto brokers, connected via communication links. Figure 1 visually represents the message flow in the system, showing how the load balancer receives and distributes publish-subscribe requests from JMeter clients to different brokers. This setup illustrates the data routing and load distribution process across brokers. All components are deployed on a single physical host to ensure consistency and reduce potential network interference.

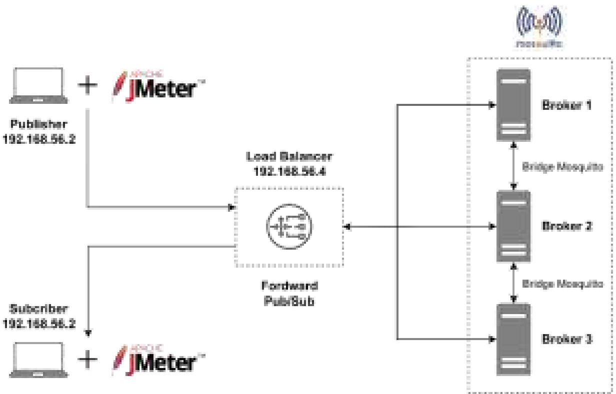


Figure 1. Proposed System Architecture

https://doi.org/10.25077/jnte.v13n3.1260.2024

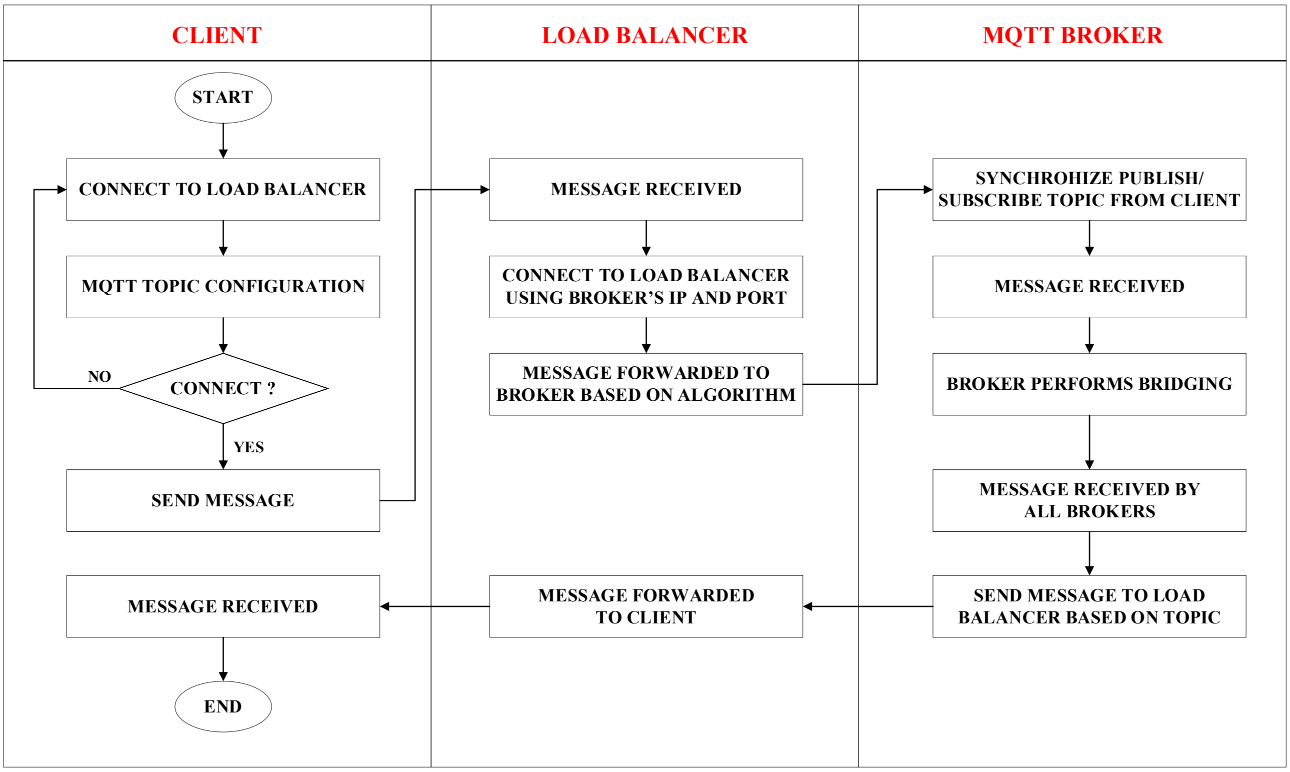


Figure 2. Proposed System Workflow

The load balancer, implemented using NGINX [26], acts as the intermediary between clients and brokers. It is configured to listen on port 1883 and handles both publish and subscribe requests from clients. For this experiment, the client configuration includes a publisher-to-subscriber ratio of 50:50, with each client generating messages at one-second intervals to simulate balanced traffic loads. The number of clients varies from 500 to 2,500, with metrics such as throughput and latency evaluated at each stage.

Apache JMeter is used to simulate traffic, generating messages that pass through the load balancer before being forwarded to the brokers. The brokers, running Mosquitto MQTT, store and synchronize incoming messages using a bridging mechanism. This ensures that all brokers maintain consistent data by forwarding messages among one another. Also, the load balancer assigns brokers to handle publish or subscribe requests based on the selected load balancing algorithm.

When a subscribe request matches a topic of interest, the corresponding message is delivered to the subscriber through the load balancer. Figure 2 provides a visual representation of the communication flow among the components.

Round Robin Load Balancing Algorithm

The RR (Round Robin) algorithm plays a critical role in efficiently processing publish and subscribe requests by ensuring each message or request is verified against prior transmissions and then forwarded to the next broker in the sequence. This approach, illustrated in Figure 3, incorporates an MQTT broker integrated with an NGINX service to handle message routing and delivery over port 1883 [27]. This setup is designed to balance the workload across multiple brokers, promoting optimal resource utilization and enhancing system performance and reliability.

The process begins with the initialization phase, where the NGINX service is launched to listen on port 1883, establishing a unified communication endpoint. This setup facilitates the seamless transmission and reception of MQTT messages, enabling both publishers and subscribers to connect through a shared entry point.

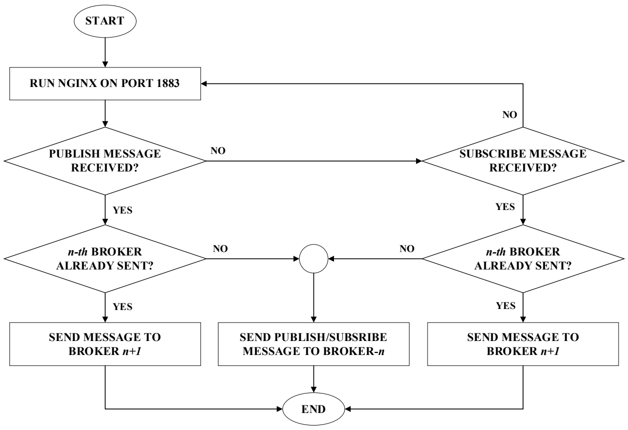


Figure 3. Round Robin Workflow

When a publish message is received from a client, the system undertakes a series of steps to handle the message efficiently. First, it verifies whether the message has already been sent to the designated broker in a previous attempt. If this is the case, the system skips re-sending it to avoid redundancy and ensure data consistency. Otherwise, the message is forwarded to the appropriate broker for processing. This verification and forwarding mechanism helps to prevent duplication, optimizing the system's resources and maintaining the accuracy of the data flow.

https://doi.org/10.25077/jnte.v13n3.1260.2024

A similar process is followed when handling subscription requests. Upon receiving a subscription request, the system checks if the request has already been forwarded to the designated broker. If the subscription has not been processed previously, it is transmitted to the broker. If it has, the system bypasses the re-transmission to prevent redundant operations, which further contributes to efficiency.

The entire workflow concludes when all publish messages and subscription requests are processed without duplication. This structured process, combining NGINX service with the RR algorithm, ensures that workloads are evenly distributed, thereby reducing latency and preventing bottlenecks. As a result, the system can maintain reliable communication between clients and brokers, promoting smooth, efficient, and consistent data exchange.

Least Response Time Algorithm

The LRT (Least Response Time) algorithm enhances the efficiency of processing publish and subscribe requests by dynamically selecting the broker with the fewest active connections or the fastest response time. Illustrated in Figure 4, this approach integrates an MQTT broker with an NGINX service operating over port 1883 [28]. Unlike the Round Robin (RR) algorithm, which distributes requests sequentially to ensure fair load distribution, the LRT algorithm distributes workloads based on real-time performance metrics, such as connection load and response time. This strategy enables optimal broker selection, directing traffic toward the most responsive broker and thereby minimizing latency while supporting scalable system performance under varying load conditions.

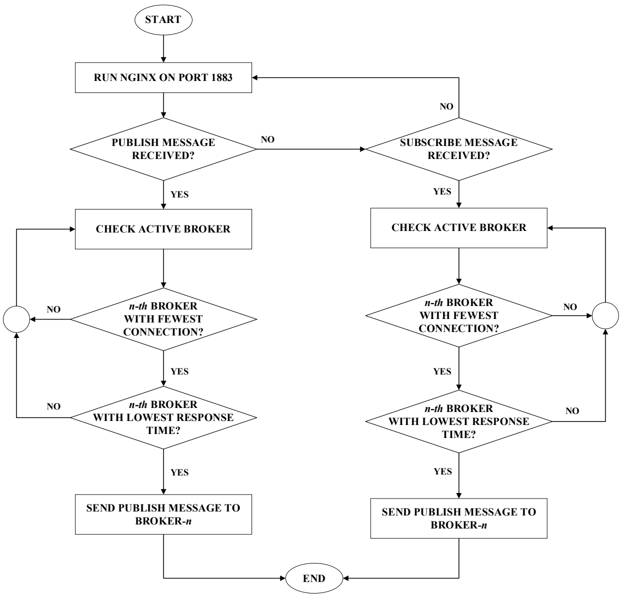


Figure 4. Least Response Time Workflow

The process begins similarly to the Round Robin approach, with an initialization phase in which the NGINX service listens on port 1883, creating a unified communication endpoint for both publishers and subscribers. This setup ensures efficient and streamlined message handling across the system.

When a publish message is detected, the system evaluates the connection load to determine the optimal broker for message forwarding. It first checks if any broker currently has the fewest active connections. If such a broker is identified, the system forwards the message to that broker for processing. However, if multiple brokers have an equal number of active connections, the system then evaluates the response times, selecting the broker with the lowest response time to ensure faster message delivery. This selection process based on real-time metrics enhances the speed and accuracy of message handling.

Similarly, when handling subscription requests, the system applies a methodical process to allocate requests to brokers efficiently. It begins by checking the connection load to identify the broker with the fewest active connections. If a broker with minimal load is available, the subscription request is forwarded to that broker. When multiple brokers have the same load, the system evaluates their response times and chooses the fastest one, ensuring minimal delay in processing the subscription request.

The entire workflow concludes after all publish messages and subscription requests have been processed efficiently, with each broker receiving the appropriate messages without duplication. By leveraging real-time performance metrics like connection load and response time, this algorithm ensures optimal workload distribution, reducing latency and preventing potential bottlenecks. This dynamic broker selection method thereby supports high system efficiency and reliability.

Experimental Setup

To evaluate the performance of the two algorithms, a series of testing scenarios were carefully designed to simulate diverse operational conditions. Each algorithm was assessed across three distinct Quality of Service (QoS) levels to capture a range of reliability requirements. At QoS 0, messages are transmitted without acknowledgment, which minimizes overhead but offers no delivery guarantee. QoS 1 adds reliability by requiring acknowledgments to confirm delivery, though this may result in potential duplicates. QoS 2 employs a two-step acknowledgment process, ensuring that each message is delivered precisely once, thus providing the highest level of reliability [29].

Testing scenarios also included five client configurations, ranging from 500 to 2,500 clients in increments of 500. To simulate real-world usage, each configuration maintained an equal split between publishers and subscribers, ensuring a balanced 50:50 ratio across all test runs. For consistency, each scenario was executed over a duration of 15 seconds, with message sizes fixed at 1,024 bytes (1 KB), allowing for straightforward comparisons across different setups.

The performance of both algorithms was then assessed using several key metrics [30], [31], [32] : CPU Usage, Throughput, Delay, Jitter, and Response Time. CPU Usage provided insights into each algorithm's efficiency by monitoring resource consumption during message processing under varying loads. Throughput was measured as the number of messages successfully processed per second, reflecting each system's capacity to handle message traffic effectively. Delay, representing the average time taken for a message to travel from the publisher to the subscriber—including processing time at the broker—was measured with Wireshark to account for the end-to-end journey. Jitter was also measured using Wireshark, focusing on the average variation in arrival times between packets to

https://doi.org/10.25077/jnte.v13n3.1260.2024

capture any fluctuations in packet delivery timing across different QoS levels. Finally, Response Time was analyzed to evaluate the system's responsiveness to client requests, providing a crucial indicator of each algorithm's real-time performance capabilities.

RESULTS AND DISCUSSION

CPU Usage Analysis

As presented in Table 1, the results highlight significant differences in CPU utilization between the RR and LRT algorithms, especially across different QoS levels and client loads. A notable trend is that LRT incurs higher CPU consumption as client load and QoS level increase, reflecting the algorithm's focus on response time optimization. In contrast, RR maintains more stable CPU usage but lacks the flexibility to adapt to real-time load variations across brokers.

At lower client loads (e.g., 500 clients), LRT efficiently distributes workload across brokers, resulting in moderate CPU usage. However, as the number of clients rises to 2000 and 2500, LRT shows a significant increase in CPU usage on specific brokers, such as broker 1 reaching 41.6% under QoS 2, compared to RR's more balanced usage pattern, peaking at 27.3% on the same broker. This suggests that LRT's dynamic selection can lead to uneven load distribution, creating the potential for bottlenecks on individual brokers under heavy traffic.

The higher CPU usage under LRT is particularly evident at QoS 1 and QoS 2, where acknowledgment overhead increases processing demands. Meanwhile, RR distributes the load evenly across brokers, making it more suitable for systems that require consistent resource utilization. However, the trade-off is that RR may introduce higher latency, as it does not prioritize response time when selecting brokers.

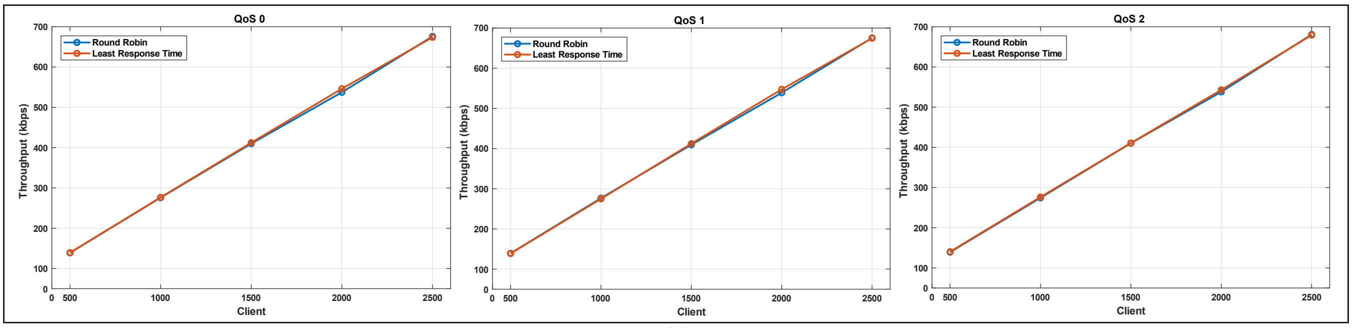
These results highlight a trade-off between response time and resource predictability. LRT is well-suited for latency-sensitive applications, where fast responses are critical, but at the cost of higher CPU consumption. In contrast, RR offers more predictable CPU usage, making it a better fit for environments with limited resources or where balanced load distribution is prioritized over latency optimization.

Throughput Analysis

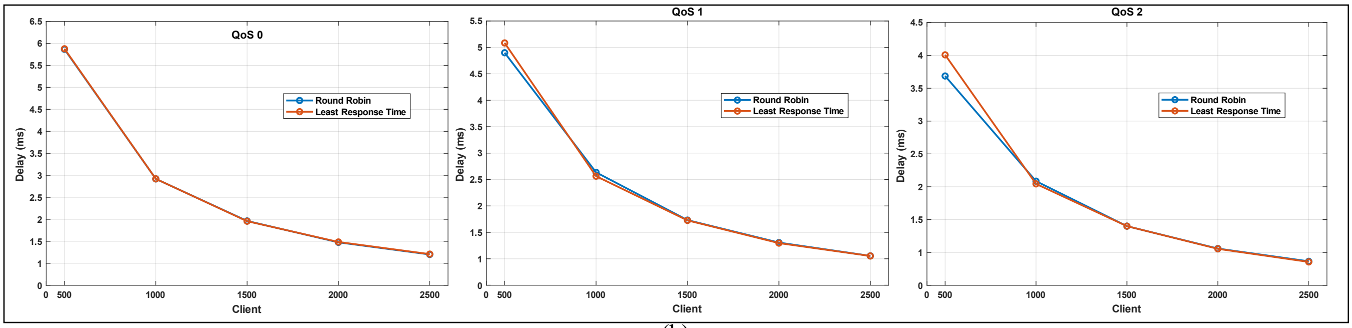
The throughput results, as shown in the Figure 5a, reveal a consistent and linear increase in message processing capability for both Round Robin (RR) and Least Response Time (LRT) algorithms across all QoS levels and client configurations. This linear behavior suggests that both algorithms are able to efficiently handle the increasing client load without a noticeable performance drop.

Table 2. CPU Usage (%)

Client	Broker	QoS 0		QoS 1		QoS 2	
		RR	LRT	RR	LRT	RR	LRT
500	1	15.7	4.8	9.8	5.2	5.3	10.4
	2	3.3	5.5	6.5	0.7	7.9	6.0
	3	6.3	6.0	6.3	6.5	5.4	1.9
1000	1	3.8	9.0	7.7	12.1	10.6	5.7
	2	10.9	12.3	10.9	6.9	14.3	16.6
	3	18.0	15.7	13.3	14.6	19.9	13.4
1500	1	10.3	24.2	9.1	20.7	21.6	21.7
	2	17.8	14.7	15.4	18.3	18.8	23.7
	3	23.0	15.6	31.4	20.2	28.2	14.3
2000	1	10.2	31.8	11.7	19.8	20.9	41.6
	2	20.4	37.1	17.3	24.3	28.7	41.2
	3	35.4	19.8	34.2	28.1	42.4	28.7
2500	1	21.1	23.2	12.5	41.6	27.3	40.7
	2	19.0	29.9	26.9	42.0	28.5	39.7
	3	51.2	32.1	44.3	25.6	60.4	44.0



(a)



(b)

Figure 5. Performance metrics of RR and LRT algorithms across different QoS levels and client loads: (a) Throughput (b) Delay

Both algorithms demonstrate almost identical throughput at QoS 0, where messages are sent without acknowledgment. As the number of clients increases from 500 to 2,500, throughput scales proportionally, confirming that the system can handle larger workloads smoothly. This indicates that neither RR nor LRT introduces overhead that would negatively impact throughput when the QoS requirement is minimal.

The QoS 1 results also show no significant divergence between the two algorithms. Both maintain high throughput, even with the added overhead of acknowledgment mechanisms required for at least once delivery. Similarly, at QoS 2, which involves a more complex two-step acknowledgment process for exactly-once delivery, throughput remains stable for both algorithms. This consistency reflects the system's ability to process messages efficiently, regardless of the acknowledgment demands introduced by higher QoS levels.

Although LRT introduces a dynamic broker selection mechanism based on response time and connection load, this feature does not result in a measurable difference in throughput compared to the simpler, sequential selection of RR. The graphs indicate that both algorithms deliver comparable throughput performance, with no apparent advantage in message processing speed under any QoS condition.

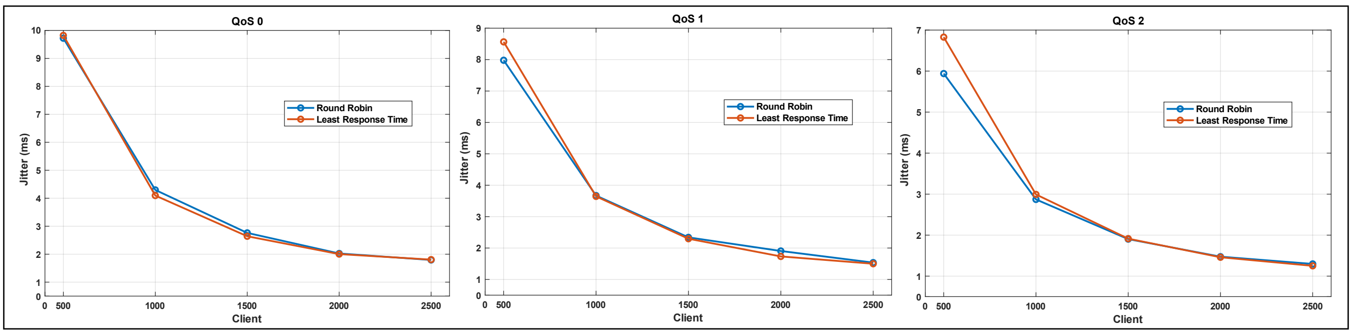
These results imply that throughput is not a limiting factor for either algorithm, making them equally effective in scenarios where high message rates are critical. The selection between RR and LRT may depend more on other factors such as response time, jitter, or CPU usage, especially in systems where latency or computational efficiency is of greater importance.

Delay Analysis

As illustrated in Figure 5b, the delay results show the time taken for messages to travel from the publisher to the subscriber under various QoS levels and client loads. RR and LRT algorithms demonstrate similar trends, with delay decreasing as the number of clients increases. However, subtle differences between the two algorithms can be observed under certain conditions.

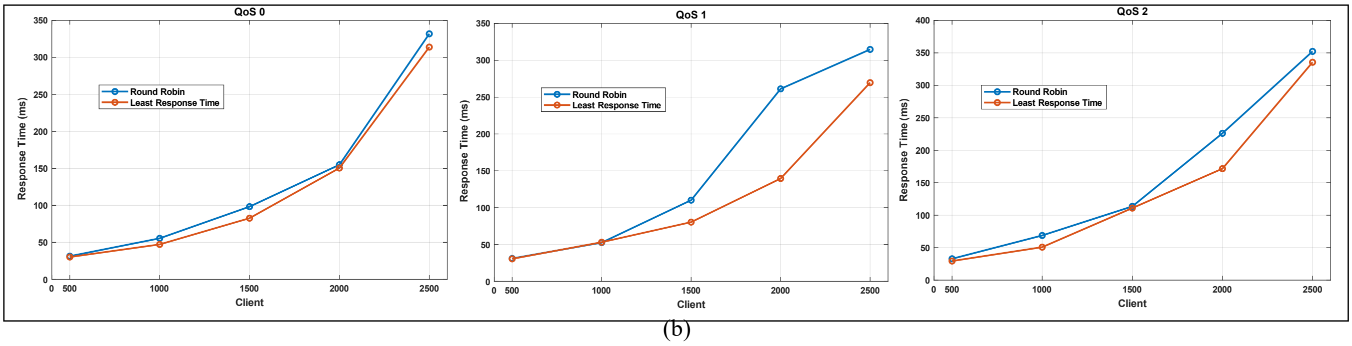
At QoS 0, where messages are delivered without acknowledgment, both algorithms exhibit an initial higher delay at 500 clients—around 5 ms—which decreases steadily to approximately 1.5 ms as the client load reaches 2,500 clients. The slight improvement in delay with increasing client numbers suggests that the system becomes more efficient in managing traffic as load increases, likely due to improved broker utilization. The general trend remains consistent in the QoS 1 and QoS 2 scenarios, with delay decreasing as more clients connect. However, the overhead introduced by acknowledgments in QoS 1 and the two-step acknowledgment process in QoS 2 slightly impacts performance. For instance, in the QoS 2 scenario at 500 clients, RR shows a marginally higher delay than LRT, indicating that LRT's adaptive broker selection may help minimize delays when acknowledgment overhead is higher.

While the differences between the two algorithms are subtle, LRT shows a small but consistent advantage in reducing delay under higher QoS levels. This is particularly evident at higher client loads, where LRT's dynamic allocation to the least-loaded broker ensures minimal delays. However, the differences are small enough that both algorithms perform effectively, especially as client numbers increase beyond 1,500 clients.



(a)

https://doi.org/10.25077/jnte.v13n3.1260.2024



(b)

Figure 6. Performance metrics of RR and LRT algorithms across different QoS levels and client loads: (a) Jitter (b) Response Time.

These results suggest that RR and LRT can handle increasing loads efficiently, improving delay as client numbers rise. LRT offers a slight advantage in scenarios with acknowledgment overhead by dynamically optimizing broker selection, making it a suitable choice for applications where low latency is critical. Meanwhile, RR provides comparable performance, particularly for systems prioritizing predictability and balanced resource allocation over minimal delay.

Jitter Analysis

As shown in Figure 6a, the jitter results measure the variability in message delivery times between the publisher and subscriber under varying QoS levels and client loads. RR and LRT algorithms exhibit a similar trend, with jitter decreasing as the number of clients increases. This suggests the system stabilizes with higher loads, likely due to more efficient resource utilization across brokers.

At QoS 0, where no acknowledgment is required, the jitter is initially higher at 500 clients—around 7 to 9 ms—for both algorithms. However, as client numbers increase to 2500, the jitter drops to approximately 1 ms, indicating that both algorithms effectively minimize delivery time fluctuations at higher loads.

Under QoS 1 and QoS 2, jitter follows a similar downward trend. However, the added acknowledgment processes in QoS 1 and the two-step acknowledgment in QoS 2 introduce slight variations. In the QoS 2 scenario, LRT shows slightly lower jitter than RR as client numbers exceed 2000, suggesting that LRT's adaptive broker selection helps maintain more consistent delivery times under higher loads.

The results highlight that, while both algorithms perform similarly at smaller client loads, LRT shows a small advantage in reducing jitter under more demanding conditions. This makes LRT more suitable for applications where predictable delivery times are critical, such as real-time communication systems. Meanwhile, RR maintains stable jitter performance across all configurations, offering a reliable solution for systems where even load distribution is more important than minimizing jitter.

Response Time Analysis

As depicted in Figure 6b, the response time results reveal how efficiently RR and LRT algorithms handle client requests under varying QoS levels and client loads. Response time, measured as the elapsed time between a client request and the system's response, provides critical insights into the system's ability to handle increasing workloads with minimal delays.

At QoS 0, both algorithms show comparable performance under lower client loads. However, as the number of clients rises beyond 1,500, RR exhibits higher response times, with a noticeable increase at 2,500 clients, reaching approximately 300 ms. In contrast, LRT maintains a lower response time under the same conditions, demonstrating its effectiveness in dynamically allocating traffic to less-loaded brokers. This suggests that LRT's adaptive approach allows it to handle large-scale requests more efficiently.

The QoS 1 and QoS 2 scenarios, the performance gap between RR and LRT becomes more apparent. With the added overhead of acknowledgment mechanisms, RR's response time continues to climb significantly, exceeding 350 ms at 2,500 clients under QoS 1. Meanwhile, LRT maintains a more moderate increase, peaking at around 250 ms under the same conditions. This consistent difference highlights LRT's advantage in reducing response times, particularly in scenarios where acknowledgments introduce additional processing demands.

The results indicate that as client loads increase, RR's sequential broker allocation can lead to delays, as it does not account for individual brokers' current load or responsiveness. In contrast, LRT's dynamic broker selection allows it to respond faster by distributing requests to brokers with the least load or fastest response, ensuring better scalability and reduced delays under high traffic.

The findings indicate that LRT (Least Response Time) is better suited for environments where minimizing response times is a priority, particularly in real-time or latency-sensitive applications. On the other hand, RR (Round Robin) may be more appropriate for systems that value fair load distribution and predictable resource usage, as they can tolerate slightly higher response times during heavy loads.

Comparative Analysis with Recent Studies

The findings of this study validate key theoretical assumptions regarding adaptive load balancing and sequential load distribution, aligning with recent literature on MQTT broker optimization. For instance, Jutadhamakorn et al. demonstrated that clustering MQTT brokers on low-cost devices like Raspberry Pi can significantly improve load distribution and scalability. This supports our findings that LRT, as an adaptive load-balancing algorithm, minimizes response time and delay through real-time broker selection, particularly under high client loads. In our study, LRT consistently outperformed RR in maintaining lower response times and delays, which aligns with the adaptive strategies observed in [33]

Similarly, Kanellopoulos and Sharma emphasized the importance of dynamic load balancing within IoT networks, noting that adaptive algorithms are essential for maintaining low latency and high throughput in real-time applications. Our findings mirror this, as LRT demonstrates better responsiveness under fluctuating client loads, making it suitable for applications that require quick response times. Conversely, Round Robin's sequential approach ensures predictable performance by balancing resource usage across brokers, confirming its suitability for environments prioritizing stability over speed. This finding aligns with[34], who observed that static algorithms like RR can maintain steady resource utilization, though they may experience higher response times under heavy loads.

Moreover, this study highlights the critical role of Quality of Service (QoS) levels in shaping MQTT system performance. Our results reveal that higher QoS levels introduce additional processing overhead, affecting both delay and jitter, a finding supported by Detti et al., who documented the impact of QoS levels on latency and reliability in MQTT networks. This underscores the need for careful QoS selection based on specific application requirements. In line with [35], our results show that RR and LRT exhibit distinct performance patterns across varying QoS levels, emphasizing the importance of aligning load-balancing strategies with QoS requirements.

Grounding this evaluation in core performance metrics—CPU usage, throughput, delay, jitter, and response time—this study bridges the gap between theoretical concepts and practical applications. Consistent with the work of [7] and [10], our analysis highlights jitter and response time as crucial metrics for real-time IoT applications, especially under heavy client loads. By examining both algorithms across multiple QoS levels, our study expands on prior research by offering a comprehensive view of each algorithm's strengths and limitations.

Situating our findings within the broader context of recent studies, this analysis provides actionable guidance for system design choices in MQTT broker performance. It confirms the trade-offs between latency and resource utilization identified in related research. It underscores the need for tailored load-balancing strategies that align with the unique demands of IoT systems and other real-time communication platforms.

CONCLUSIONS

This study compares Round Robin (RR) and Least Response Time (LRT) algorithms for optimizing MQTT broker performance. The results highlight that LRT enhances response time and reduces delay, especially under high client loads and QoS levels, though it incurs higher CPU usage. Meanwhile, RR ensures balanced CPU utilization and predictable performance but introduces slightly higher response times under heavy workloads.

Both algorithms exhibit linear scalability, confirming their capability to handle growing workloads. However, LRT is more suitable for real-time applications, while RR fits resource-constrained environments requiring stability. LRT can be particularly advantageous in real-world applications in latency-sensitive scenarios, such as industrial automation or telemedicine, where rapid response times are critical. On the other hand, RR may be more applicable in IoT settings with limited processing capacity, such as environmental monitoring systems, where even load distribution is prioritized over low latency.

A key challenge identified is balancing low latency with resource efficiency, especially under varying network conditions. Future research could explore hybrid algorithms that combine the strengths of RR and LRT to optimize performance in MQTT-based systems further.

ACKNOWLEDGMENT

This research was financially supported by the Lembaga Penelitian dan Pengabdian Masyarakat (LPPM) of Institut Teknologi dan Bisnis Asia Malang through an Internal Research Grant Year 2024.

REFERENCES

[1] H. S. Pramono, V. G. Al Hakim, and F. Alfianto, “IoT-Based Disaster Response Robot for Victim Identification in Building Collapses,” *Jurnal Nasional Teknik Elektro*, pp. 51–59, Mar. 2024, doi: 10.25077/jnte.v13n1.1191.2024.

[2] M. A. Afandi, I. Hikmah, and C. Agustinah, “Microcontroller-based Artificial Lighting to Help Growth the Seedling Pakcoy,” *JURNAL NASIONAL TEKNIK ELEKTRO*, vol. 10, no. 3, Nov. 2021, doi: 10.25077/jnte.v10n3.943.2021.

[3] Rocky Alfanz, A. H. A. Aqbal, and Wahyuni Martiningsih, “Smart Farm Agriculture Design by Applying a Solar Power Plant,” *JURNAL NASIONAL TEKNIK ELEKTRO*, Jul. 2023, doi: 10.25077/jnte.v12n2.1085.2023.

[4] Z. Zickri, A. Novandri, R. Adriman, and Nasaruddin, “Shrimp Pond Monitoring System using Cooperative Wireless Sensor Network Multi-Hop Technique based on Internet of Things,” *Jurnal Nasional Teknik Elektro*, pp. 56–63, Dec. 2023, doi: 10.25077/jnte.v12n3.1133.2023.

[5] P. D. P. Adi *et al.*, “Application of IoT-LoRa Technology and Design in irrigation canals to improve the quality of agricultural products in Batu Indonesia,” in *2021 2nd International Conference On Smart Cities, Automation & Intelligent Computing Systems (ICON-SONICS)*, IEEE, Oct. 2021, pp. 88–94. doi: 10.1109/ICON-SONICS53103.2021.9617175.

[6] P. D. Prasetyo Adi *et al.*, “ECG-LPWAN based for Real-time monitoring Patient's Heart Beat Status,” in *2022 International Seminar on Application for Technology of Information and Communication (iSemantic)*, IEEE, Sep. 2022, pp. 7–14. doi: 10.1109/iSemantic55962.2022.9920379.

[7] T. Domínguez-Bolaño, O. Campos, V. Barral, C. J. Escudero, and J. A. García-Naya, “An overview of IoT

architectures, technologies, and existing open-source projects," *Internet of Things*, vol. 20, p. 100626, Nov. 2022, doi: 10.1016/j.iot.2022.100626.

[8] L. Nurfiqin, Z. Sari, F. Dwi, and S. Sumadi, "Analisis Quality of Service (QoS) Protokol MQTT dan HTTP Pada Sistem Smart Metering Arus Listrik," *REPOSITOR*, vol. 3, no. 1, pp. 121–130, 2021.

[9] M. Amoretti, R. Pecori, Y. Protskaya, L. Veltri, and F. Zanichelli, "A Scalable and Secure Publish/Subscribe-Based Framework for Industrial IoT," *IEEE Trans Industr Inform*, vol. 17, no. 6, pp. 3815–3825, Jun. 2021, doi: 10.1109/TII.2020.3017227.

[10] A. P.S., S. M. Dilip Kumar, and V. K.R., "MQTT Implementations, Open Issues, and Challenges: A Detailed Comparison and Survey," *International Journal of Sensors, Wireless Communications and Control*, vol. 12, no. 8, pp. 553–576, Oct. 2022, doi: 10.2174/2210327913666221216152446.

[11] K. Charlie, R. Primananda, and M. Data, "Analisis Performa Load Balancing Pada Broker MQTT Menggunakan Algoritma Round Robin," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer* , vol. 2, no. 9, pp. 2902–2908, 2018, [Online]. Available: http://j-ptiik.ub.ac.id

[12] F. Montori, L. Gigli, L. Sciullo, and M. Di Felice, "LA-MQTT: Location-aware Publish-subscribe Communications for the Internet of Things," *ACM Transactions on Internet of Things*, vol. 3, no. 3, pp. 1–28, Aug. 2022, doi: 10.1145/3529978.

[13] E. Longo, A. E. C. Redondi, M. Cesana, and P. Manzoni, "BORDER: A Benchmarking Framework for Distributed MQTT Brokers," *IEEE Internet Things J*, vol. 9, no. 18, pp. 17728–17740, Sep. 2022, doi: 10.1109/JIOT.2022.3155872.

[14] P. Colombo, E. Ferrari, and E. D. Tümer, "Regulating data sharing across MQTT environments," *Journal of Network and Computer Applications*, vol. 174, p. 102907, Jan. 2021, doi: 10.1016/j.jnca.2020.102907.

[15] F. Sisilia Mukti and F. E. Nugroho, "Upaya Pencapaian Status High Availability Server Menggunakan Metode Load Balancing Berbasis Klaster pada Database Server PT. XYZ," *Jurnal Teknik Informatika dan Sistem Informasi (JATISI)*, vol. 10, no. 1, pp. 2407–4322, 2023, doi: https://doi.org/10.35957/jatisi.v10i1.5886.

[16] H. Anwer, F. Azam, M. W. Anwar, and M. Rashid, "A Model-Driven Approach for Load-Balanced MQTT Protocol in Internet of Things (IoT)," 2020, pp. 368–378. doi: 10.1007/978-3-030-22354-0_33.

[17] M. G. Spina, G. M. Marotta, S. Gualtieri, and F. De Rango, "Topic Load Balancing in a multi IoT Gateways Scenario under Publish/Subscribe Paradigm," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, Jan. 2022, pp. 521–522. doi: 10.1109/CCNC49033.2022.9700606.

[18] H. Triangga, I. Faisal, and I. Lubis, "Analisis Perbandingan Algoritma Static Round-Robin dengan Least-Connection Terhadap Efisiensi Load Balancing pada Load Balancer Haproxy," *InfoTekJar (Jurnal Nasional Informatika dan Teknologi Jaringan)*, vol. 4, no. 1, pp. 70–75, Sep. 2019, doi: 10.30743/infotekjar.v4i1.1688.

[19] M. A. Spohn, "On MQTT Scalability in the Internet of Things: Issues, Solutions, and Future Directions," *Journal of Electronics and Electrical Engineering*, p. 4, Oct. 2022, doi: 10.37256/jeee.1120221687.

[20] K. Charlie, R. Primananda, and M. Data, "Analisis Performa Load Balancing Pada Broker MQTT Menggunakan Algoritma Round Robin," *Jurnal Pengembangan Teknologi Informasi dan Komputer*, vol. 2, no. 9, pp. 2902–2908, 2018, [Online]. Available: http://j-ptiik.ub.ac.id

[21] Sepriano, V. Suryani, and E. M. Jadied, "Analisis Performansi Metode Load Balancing Pada Broker Protokol Mqtt Menggunakan Algoritma Least Connection (2021)," in *e-Proceeding of Engineering* , 2021, pp. 2936–2951.

[22] J. Li, "Research and Application of Dynamic Load Balancing in MQTT Server," *Software Engineering and Applications*, vol. 09, no. 04, pp. 262–271, 2020, doi: 10.12677/SEA.2020.94029.

[23] A. Naufal Romiz, E. Sakti Pramukantoro, and W. Yahya, "Implementasi Mekanisme Load Balancer dan Failover pada IoT Middleware berbasis Publish-Subscribe," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 3, no. 5, pp. 5165–5174, 2019, [Online]. Available: http://j-ptiik.ub.ac.id

[24] B. Mishra, B. Mishra, and A. Kertesz, "Stress-Testing MQTT Brokers: A Comparative Analysis of Performance Measurements," *Energies (Basel)*, vol. 14, no. 18, p. 5817, Sep. 2021, doi: 10.3390/en14185817.

[25] A. Et al., "Developing Load Balancing for IoT - Cloud Computing Based on Advanced Firefly and Weighted Round Robin Algorithms," *Baghdad Science Journal*, vol. 16, no. 1, p. 0130, Mar. 2019, doi: 10.21123/bsj.2019.16.1.0130.

[26] F. Apriliansyah, I. Fitri, and A. Iskandar, "Implementasi Load Balancing Pada Web Server Menggunakan Nginx," *Jurnal Teknologi dan Manajemen Informatika*, vol. 6, no. 1, Apr. 2020, doi: 10.26905/jtmi.v6i1.3792.

[27] D. T. Nababan, R. Primananda, and F. A. Bakhtiar, "Implementasi Load Balancing pada Broker MQTT dengan Algoritme Weighted Least Connection menggunakan Raspberry Pi," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 5, no. 3, pp. 867–875, 2021, [Online]. Available: http://j-ptiik.ub.ac.id

[28] H. K. Ramadhan and S. Wardhana, "Optimalisasi Jaringan Komputer Menggunakan Algoritma Load Balancing di Virtual Server Citrix ADC," in *Seminar Nasional Pengaplikasian Telematika (SINAPTIKA)* , 2021, pp. 135–148.

[29] F. Ilham, A. G. Putrada, and S. Prabowo, "Analisis performansi QoS MQTT pada sistem monitoring sungai," in *e-Proceeding of Engineering*, 2019, pp. 2006–2020.

[30] Y. Susdyastama Putra, M. T. Indriastuti, and S. Mukti, "Optimalisasi Nilai Throughput Jaringan Laboratorium Menggunakan Metode Hierarchical Token Bucket (Studi Kasus: STMIK Asia Malang)," *Jurnal Ilmiah NERO*, vol. 5, no. 2, pp. 83–90, 2020.

[31] D. D. Mahendra and F. S. Mukti, "Sistem Deteksi dan Pengendalian Serangan Denial of Service pada Server

Berbasis Snort dan Telegram-API," *Techno.Com*, vol. 21, no. 3, pp. 511–522, Aug. 2022, doi: 10.33633/tc.v21i3.6466.

[32] J. Toldinas, B. Lozinskis, E. Baranauskas, and A. Dobrovolskis, "MQTT Quality of Service versus Energy Consumption," in *2019 23rd International Conference Electronics*, IEEE, Jun. 2019, pp. 1–4. doi: 10.1109/ELECTRONICS.2019.8765692.

[33] P. Jutadhamakorn, T. Pillavas, V. Visoottiviseth, R. Takano, J. Haga, and D. Kobayashi, "A Scalable and Low-Cost MQTT Broker Clustering System," in *2nd International Conference on Information Technology (INCIT)*, IEEE, 2017.

[34] D. Kanellopoulos and V. K. Sharma, "Dynamic Load Balancing Techniques in the IoT: A Review," Dec. 01, 2022, *MDPI*. doi: 10.3390/sym14122554.

[35] A. Detti, L. Funari, and N. Blefari-Melazzi, "Sub-Linear Scalability of MQTT Clusters in Topic-Based Publish-Subscribe Applications," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1954–1968, Sep. 2020, doi: 10.1109/TNSM.2020.3003535.

AUTHOR(S) BIOGRAPHY

Samsul Arifin received a bachelor's degree in Information Technology from Politeknik Negeri Surabaya and a master's degree in Information Technology from Institut Sains dan Teknologi Terpadu Surabaya, and is currently pursuing a doctoral degree at Universitas Negeri Malang. His research interests are intelligent control algorithms and robotics.

Alfian Wahyu Nugraha earned his bachelor's degree in Informatics Engineering from Institut Asia Malang. His research focuses on computer networks, IoT and mobile application. He has two years of experience as a network and computer lab assistant, where he supported practical learning and network infrastructure management. Currently, he is part of the academic team at Institut Asia Malang, actively contributing to coordinate academic activities.

Fransiska Sisilia Mukti was born in Malang, Indonesia, on September 28, 1990. She earned her bachelor degree from the Department of Electrical Engineering, Institut Teknologi Nasional Malang, in 2013. She completed her master's degree in Electrical Engineering at Universitas Brawijaya in 2018. Currently, she is pursuing her doctoral in Electrical Engineering at Institut Teknologi Sepuluh Nopember, Surabaya. Her research focuses on computer networks, propagation, IoT, cybersecurity, and artificial intelligence. She works as a lecturer and serves as the Head of Network & Computer Infrastructure at Institut Asia Malang. She can be contacted via email: ms.frans@asia.ac.id.

Sunu Jatmika earned his bachelor's degree in Informatics and Computer Engineering from STIKI Malang and his master's degree in Information Technology from STTS Surabaya in 2014. He is currently pursuing a doctoral degree at Universitas Negeri Malang. His research focuses on mobile programming, Android development, and business intelligence. He actively contributes to academic and research activities, developing innovative solutions for mobile and business applications.

 https://doi.org/10.25077/jnte.v13n3.1260.2024