Evaluating Service Quality in NGINX Load Balancing: A Comparative Study of Round Robin and Least Connection Algorithms

Muhammad Aldy Ikramsyah[1], Henki Bayu Seta[2*], Ika Nurlaili Isnainiyah [3], Theresiawati [4],
Computer Science Faculty, Universitas Pembangunan Nasional Veteran Jakarta, Jakarta, Indonesia

Article Info

Article history:

Received Apr 5, 2025
Revised Jul 10, 2025
Accepted Jul 10, 2025

Keywords:

Service Quality
Load Balancing
Round Robin
Least Connection

ABSTRACT

This study evaluates the Quality of Service (QoS) of the Round-Robin and Least Connection algorithms in load balancing with NGINX. Round Robin outperformed and was consistent across most of the evaluated parameters, including throughput, latency, jitter, and packet loss. Its simplicity stems from the repeated distribution of requests without monitoring server connections, making it ideal for homogeneous traffic and consistent server specifications. However, its performance is poor in dynamic traffic or heterogeneous server architectures. In contrast, Least Connection is more advanced in accommodating dynamic traffic and heterogeneous server environments by allocating load in real-time based on active server connections. Nevertheless, Least Connection showed instability in various tests, especially at high thread counts, where anomalies such as substantial value spikes were recorded. This indicates that it is not suitable for scenarios involving uniform traffic and server specifications. Least Connection is more suitable for complex networks with dynamic traffic and varying server specifications, while Round Robin is recommended for stable workload environments because it provides consistency and simplicity. Round Robin demonstrated superior overall performance, while Least Connection excelled in adaptability, highlighting the need to align algorithm selection with traffic characteristics and server infrastructure. To improve the effectiveness of load balancing strategies, future research should investigate dynamic traffic, higher loads, and diverse environments.

Corresponding Author:

Henki Bayu Seta
Faculty of Computer Science
Universitas Pembangunan Nasional Veteran Jakarta,
Jl. Rs. Fatmawati, Pondok Labu, Jakarta Selatan, DKI Jakarta, 12450, Indonesia
Email: henkiseta@upnvj.ac.id

I. INTRODUCTION

The development of information technology, particularly the internet, has become a major pillar in supporting the activities of modern society. Beside being a communication tool, the internet also serves as the foundation for various digital services for consumers and businesses. Fast and reliable internet connectivity has become essential in the rapidly evolving digital age [1]. However, as the number of users and service demand increase, the task of maintaining network stability becomes increasingly complex. According to the Indonesian Internet Service Providers Association (APJII), the internet penetration rate in Indonesia in 2024 is 79.5%, with 221,563,479 internet users out of the total population in 2023 [2]. The increasing number of internet users can lead to internet traffic congestion, causing various problems such as jitter, latency, server overload, and potential packet loss [3]. The congestion caused by network traffic also impacts bandwidth availability, which is shrinking as consumption increases, resulting in poor network throughput and slow network operations.

One solution to this problem is to implement load balancing, which is a technology that distributes requests or traffic across multiple servers to optimize performance and ensure uninterrupted service [4], [5]. NGINX, one of the most popular load balancing software, supports various traffic distribution algorithms, including Round Robin and Least Connections. Round Robin distributes traffic evenly among servers, while Least Connection distributes based on the number of active connections. Although these two algorithms use different approaches, their effectiveness can vary depending on traffic conditions and system configuration [6], [7].

Unfortunately, research comparing the performance of these two algorithms is still limited, particularly in terms of Quality of Service (QoS), which includes metrics such as response time, throughput, and error rate. Most research only focuses on applying a single algorithm without investigating the benefits and drawbacks of each algorithm in different scenarios [8]. This results in a knowledge gap regarding the optimal algorithms to apply in non-uniform server infrastructure or dynamic traffic conditions.

This study aims to bridge this gap by analyzing and comparing the QoS of the Round Robin and Least Connection algorithms on NGINX. The study uses load testing to evaluate the performance of both algorithms under various traffic scenarios, including uniform and non-uniform traffic distribution. The collected data will be used to provide insights into the optimal conditions for each algorithm and how these conditions affect the stability and performance of the network. The findings of this study are expected to help network administrators select and configure load balancing algorithms that meet their requirements. Understanding the strengths and limitations of both algorithms allows for more targeted solutions to improve the quality of network services for individual and business users.

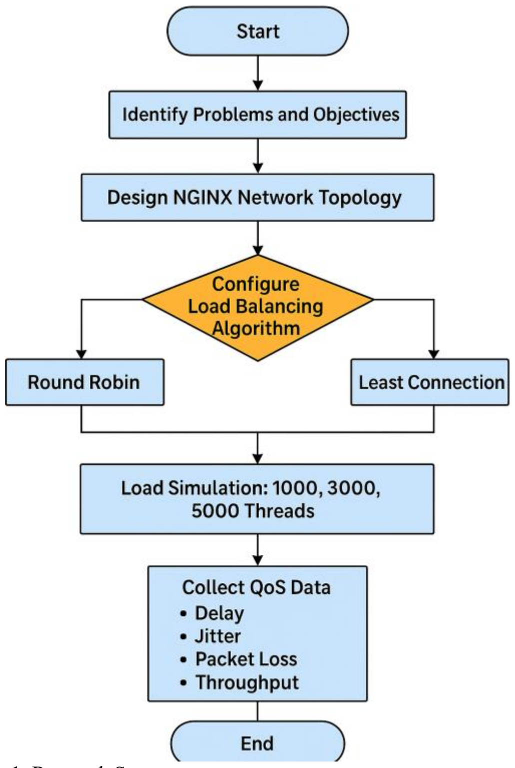
II. METHODOLOGY

This study uses a controlled experimental method to test QoS in the implementation of the Round Robin and Least Connection algorithms on NGINX load balancing. The research stages can be seen in Figure 1. The testing environment will be designed to provide consistent conditions for both algorithms. NGINX load balancing will be configured to distribute traffic across a number of backend servers, each of which will have comparable specifications to ensure fair testing conditions. The Round Robin and Least Connection algorithms will be implemented independently and evaluated under low, medium, and high load conditions to assess their performance at various request levels.

The first step in this research is to create a network topology suitable for testing the algorithm. This topology involves configuring NGINX load balancing to direct traffic to multiple backend servers. To ensure consistent testing conditions, all servers will have the same specifications. This topology design is important to ensure that the testing environment is ready to implement and evaluate the performance of the two load balancing algorithms that will be tested. At this stage, the load balancing algorithms are applied to the web servers thru NGINX configuration. The Round Robin and Least Connection algorithms will be run on the servers to distribute client requests evenly according to the principles of each algorithm. This setting ensures that traffic is automatically redirected to the active server based on the workload received.

After the configuration is complete, the algorithm is tested to evaluate the performance of each method. This testing requires load simulation with varying intensities (low, medium, and high). If the testing fails, the configuration will be reviewed and adjusted until the algorithm functions correctly. This test is designed to ensure that both algorithms can handle the specified traffic. This test is designed to ensure that both algorithms can handle the specified traffic.

The collected data includes response time, throughput, and error rate of requests for each algorithm across all load scenarios. This data was then automatically recorded using testing tools such as JMeter or Apache Benchmark to ensure accuracy and reliability. Data was analyzed to compare the performance of both algorithms in terms of Quality of Service. This analysis compared key metrics such as average response time, throughput, and request failure rate. This analysis identifies algorithms that are more effective and efficient in managing various workloads.


Start
Identify Problems and Objectives
Design NGINX Network Topology
Configure Load Balancing Algorithm
Round Robin
Least Connection
Load Simulation: 1000, 3000, 5000 Threads
Collect QoS Data
• Delay
• Jitter
• Packet Loss
• Throughput
End
Fig. 1. Research Stages

III. Result and Discussion

A. Network Topology Design

The author created a diagram of the HTTP request flow routed to the web server thru load balancing. This scheme is intended to facilitate effective test planning and serve as a reference for subsequent topology development. This design includes IP address allocation for load balancing and web servers. Figure 2 illustrates the flowchart of the HTTP Request testing process, which passes thru NGINX load balancing before being distributed to multiple web servers. This scheme is intended to ensure that client requests are correctly routed thru load balancing, which then distributes the load across three backend servers with different IP addresses. This approach allows for the assessment of load balancing efficiency in distributing requests fairly or according to a predefined algorithm, such as Round Robin or Least Connection.

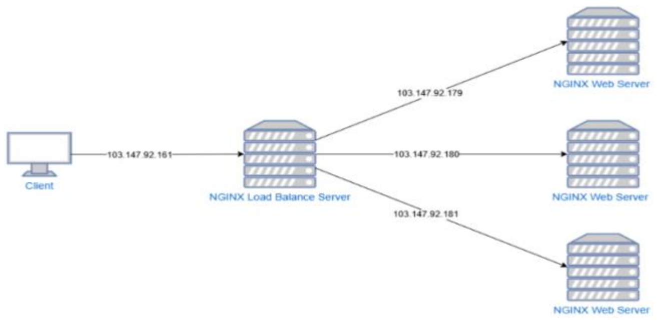


Fig. 2. Network Topology

In this scheme, the client sends an HTTP request to the NGINX load balancer at IP address 103.147.92.161. Load balancing will receive the request and distribute it to one of the three available web servers using the configured load balancing algorithm. The IP addresses of each web server are as follows: 103.147.92.179, 103.147.92.180, and 103.147.92.18. These IP addresses are used to identify each server, enabling load balancing to direct requests to the appropriate server. This design aims to facilitate realistic load testing by directing various client requests to three web servers thru load balancing. This topology allows us to measure performance metrics such as response time, throughput, and load distribution on each server. Understanding the effectiveness of load balancing algorithms in managing high demand and ensuring optimal load distribution among existing servers is crucial.

Assigning different IP addresses to load balancing and each web server is intended to simplify configuration management, both on the network side and when implementing NGINX. The main address for receiving requests from clients is the load balancing IP address (103.147.92.161), while the web server IP addresses allow load balancing to distribute the load to specific servers. This design is intended to serve as a reference in developing an appropriate topology for valid and comprehensive performance testing of load balancing algorithms.

B. QoS Round Robin Metric Calculation

Table I presents the delay calculations for the QoS Round Robin Metric using different thread counts—1000, 3000, and 5000. In the 1000-thread test, three trials yielded consistent Total Delay/s values ranging from 9.97918 to 9.984619 seconds. The average delay per second (Delay/s) was approximately 0.0099 seconds, with values ranging from 9.979180 ms to 9.994614 ms when converted to milliseconds (Delay/ms). This result shows that with 1000 threads, the system can handle requests with relatively short and consistent delays.

TABLE I. QoS Round Robin Metric Calculation – Delay

	1000 Thread		
Test number-	**Total Delay's**	**Delay/s**	**Delay/ms**
1	9,97918	0,009979	9,979180
2	9,984619	0,009995	9,994614
3	9,98291	0,009993	9,992903
	3000 Thread		
Test number-	**Total Delay's**	**Delay/s**	**Delay/ms**
1	38,06218	0,012902	12,90243
2	38,106107	0,012909	12,90857
3	38,167747	0,013000	12,99991
	5000 Thread		
Test number-	**test number-**	**test number-**	**test number-**
1	58,32613	0,012123	12,12349
2	57,860226	0,011955	11,95459
3	58,48759	0,012221	12,22056

In the second test with 3000 threads, the Total Delay/s value increased from 38.06218 to 38.167747 seconds. The Delay/s value increased to approximately 0.0129 seconds, which, when converted to milliseconds, resulted in a Delay/ms between 12.90243ms and 12.99991ms. This indicates that although the recorded delay increased with the increasing number of threads, it remained within a relatively stable range of around 12ms. In the final test involving 5000 threads, the Total Delay/s value increased to approximately 57.860226 to 58.48759 seconds. The Delay/ms value in this test ranged from 11.95459ms to 12.22056ms, with the recorded delay/s between 0.011955 and 0.012221 seconds. Despite a substantial increase in threads, the delay only experienced a marginal increase relative to testing with 3000 threads, indicating the system's proficiency in effectively distributing the load across a large number of requests.

Overall, the delay calculation results show that the system can handle an increased number of threads while maintaining relatively stable performance. Although the delay value increased with the number of threads, the difference was not significant, indicating that the load balancing algorithm used was quite effective in maintaining Quality of Service even when demand increased. Based on the jitter calculation table in Table II, with varying numbers of threads (1000, 3000, and 5000). When testing with 1000 threads, jitter is calculated based on the variation in delay between tests. The recorded delay variations ranged from 0.000598 to 0.002976 seconds, and the jitter per second (Jitter/s) was very small, ranging from 5.9918E-07 to 2.98196E-06. The jitter per millisecond (Jitter/ms) ranged from 0.000599198 ms to 0.002981964 ms after being converted to milliseconds. This result shows that at 1000 threads, jitter is low and stable, indicating that the system can maintain consistent latency between requests.

Table II. Results of QoS Round Robin Metric Calculation - Jitter In testing with 3000 threads, the delay variation increased from 0.011654 to 0.017942 seconds. The jitter in

this test ranged from 3.9707E-06 to 6.07997E-06, while the jitter in milliseconds (Jitter/ms) ranged from 0.003970698ms to 0.006079973. The increased number of threads caused a slight increase in jitter, but the value remained low, indicating that the system performed well even under increased load [9]. The variation in delays recorded in the latest testing, which involved 5000 threads, ranged from 0.000504 to 0.017221 seconds. The Jitter/s values range from 1.05307E-07 to 3.55806E-06, and the Jitter/ms values range from 0.000105307 ms to 0.003558085 ms. This result shows that, although the number of threads increased to 5000, jitter remained under control and only increased slightly. This indicates that the system can maintain consistent response times across requests even when traffic increases significantly.

TABLE II. RESULTS OF QOS ROUND ROBIN METRIC CALCULATION - JITTER

	1000 Thread	
Delay/s Variations	**Jitter/s**	**Jitter/ms**
0,002017	2,02104E-06	0,002021042
0,000598	5,99198E-07	0,000599198
0,002976	2,98196E-06	0,002981964
	3000 Thread	
Delay/s variations	**Delay/s**	**Delay/ms**
0,01402	4,75415E-06	0,004754154
0,017942	6,07997E-06	0,006079973
0,011654	3,9707E-06	0,003970698
	5000 Thread	
Delay/s variations	**Delay/s**	**Delay/ms**
0,003654	7,59509E-07	0,000759509
0,017221	3,55806E-06	0,003558058
0,000504	1,05307E-07	0,000105307

Overall, the jitter calculation results show that the load balancing system can effectively handle delay variations, even when the number of threads is increased from 1000 to 5000. Low and stable jitter indicates that the load balancing algorithm used is highly effective in maintaining QoS by minimizing delay fluctuations between requests. Ensuring a consistent user experience is crucial, especially for applications that require fast and reliable response times. Packet loss occurs when one or more data packets sent over a network do not reach their intended destination [10], [11]. This can happen for various reasons, including network overload, hardware issues, or network disruptions. Table III shows the packet loss calculation results from three trials.

TABLE III. QOS ROUND ROBIN METRIC – PACKET LOSS CALCULATION RESULTS

		1000 Thread		
Test Number	**Package Shipped**	**Package Received**	**Total Packages**	**Package Lost**
1	1000	1000	0	0%
2	1000	1000	0	0%
3	1000	1000	0	0%
		3000 Thread		
Test Number	**Package Shipped**	**Package Received**	**Total Packages**	**Package Lost**
1	3000	2950	50	1,67%
2	3000	2952	48	1,60%
3	3000	2936	64	2,13%
		5000 Thread		
Test Number	**Package Shipped**	**Package Received**	**Total Packages**	**Package Lost**
1	5000	4811	189	3,78%
2	5000	4840	160	3,20%
3	5000	4786	214	4,28%

Based on the table above, the packet loss rate for each packet in the first 1000 packets of the 1000-packet test is 0%. Similar to the initial testing, 1000 packets were sent and received without any loss. The packet loss rate is 0%. In the third test, all 1000 packets sent were received intact. The packet loss rate is 0%. As a result, there was no packet loss in the transmission of 1000 threads. Out of 3000 packages sent, 2950 were successfully delivered, leaving 50 packages whose whereabouts are unknown. The packet loss rate is 1.67%. In the second transmission, 3,000 packets were sent and 2,952 were received, resulting in a loss of 48 packets. The packet loss rate is 1.60%. In the third test, 2,936 out of 3,000 sent packets were received, resulting in a loss of 64 packets. The packet loss rate is 2.13%, with slight differences in each test and an average of about 1.8%. According to the test results involving 5000 threads, out of 5000 packets sent, only 4811 were successfully received, resulting in a loss of 189 packets, and the packet loss rate is 3.78%. In the second transmission of 5,000 packets, 4,840 were successfully received, with 160 packets not found, and the packet loss rate is 3.20%. In the final test, 5000 packets were sent, 4786 of which were received, resulting in a loss of 214 packets and a packet loss rate of 4.28%. The packet loss rate has increased significantly compared to a smaller number of threads, with an average loss of about 3.8%. As the number of threads increases, Throughput is a measure of how much data is successfully transmitted thru a system or network within a given time period [12], [13]. In computer networks, throughput indicates the effective data transmission capacity between two points [14], [15]. Table IV shows the throughput values obtained from three trials: 1000 threads, 3000 threads, and 5000 threads.

TABLE IV. RESULTS OF THROUGHPUT CALCULATION FOR ROUND ROBIN QOS METRIC

	1000 Thread		
Test Number	**Total Bit**	**Time Span(s)**	***Throughput***
1	7813344	9,979	782.978,6552
2	7813312	9,985	782.504,9574
3	7813344	9,984	782.664,9304
	3000 Thread		
Test Number	**Total Bit**	**Time Span(s)**	***Throughput***
1	23049312	38,962	605.572,8023
2	23064960	38,106	605.284,2072
3	22939904	38,168	601.024,5232
	5000 Thread		
Test Number	**Total Bit**	**Time Span(s)**	***Throughput***
1	37589936	58,326	644.479,9232
2	37816576	57,869	653.587,5562
3	37394624	58,488	639.355,4917

Table IV presents the throughput test results for various numbers of threads. Testing with 1000 threads resulted in a throughput of 782.98 kbps with a total of 7,813,344 bits and a transmission time of 9.979 seconds. Throughput decreased slightly to 782.50 kbps in the second test, which had a transmission time of 9.985 seconds and an almost identical total number of bits (7,813,312). The third test yielded comparable results, maintaining the same number of bits and a duration of 9.984 seconds, resulting in a throughput of 782.66 kbps. Throughput remained stable at around 782 kbps, indicating consistent performance with 1000 threads. Testing with 3000 threads resulted in 23,049,312 bits transmitted over

38.962 seconds, yielding a throughput of 605.57 kbps. In the second test, with a total of 23,046,960 bits and a time of 38.962 seconds, throughput decreased slightly to 605.28 kbps. The third test produced a throughput of 601.02 kbps, with a total of 22,939,904 bits transmitted in Throughput is approximately 605 kbps, showing slight fluctuations in each test, which may be due to decreasing efficiency as the number of threads increases.

The third test with 5000 threads yielded a throughput of 644.48 kbps, with a total of 37,589,936 bits and a transmission time of 58.326 seconds. In the second test, a total of 37,816,576 bits were transmitted over 57.869 seconds, resulting in a higher throughput of 653.59 kbps. The third test documented a total of 37,394,624 bits transmitted over 58.488 seconds, yielding a throughput of 639.36 kbps. Throughput varied between 639 and 653 kbps, and on average decreased compared to lower thread counts, likely due to congestion effects at higher thread counts. Average throughput decreased as the number of threads increased from 1000 to 5000. Throughput was higher and relatively stable at 1000 threads. Conversely, throughput decreased and showed more pronounced fluctuations at 3000 and 5000 threads. This may be due to the limited network capacity to accommodate the increased number of threads, leading to a decrease in data transmission efficiency at higher thread levels [16], [17].

C. Calculation of Minimum QoS Connection Metrics

Table V shows the delay calculations for the QoS metric Least Connection with different numbers of threads (1000, 3000, and 5000 threads). The calculation of the Least Connection QoS metric in Test 1 with 1000 threads resulted in a total delay of 12.99383 seconds, an average delay per second of 0.012994 seconds, and a delay per millisecond of 12.99383 ms. The second test showed a total delay of 10.98023 seconds, an average delay per second of 0.01098 seconds, and a delay per millisecond of 10.98023 ms. The third test yielded a total delay of 11.08898 seconds, with an average delay of 0.011089 seconds and a delay per millisecond of 11.08898 ms. At a load of 1000 threads, the delay was smaller and more stable, indicating that the system performed well even under low load. Testing with 3000 threads is recommended for a larger number of threads. Test 1 showed a total delay of 24.00382 seconds, an average delay per second of 0.011113 seconds, and a delay per millisecond of 11.11289 ms. The second test had a total delay of 36.22308 seconds, an average delay of 0.012304 seconds, and a delay of 12.30404 milliseconds. The third test showed a total delay of 38.5215 seconds, an average delay per second of 0.013103 seconds, and a delay per millisecond of 13.10255 ms. As the number of threads increased, so did the delay, indicating that the system was starting to experience a heavier load [18].

In the maximum load test (5000 threads), Test 1 had a total delay of 58.70857 seconds, an average delay per second of 0.012428 seconds, and a delay per millisecond of 12.42773 ms. In the second test, the total delay was 58.19657 seconds, with an average delay per second of 0.01234 seconds and a delay per millisecond of 12.34024 ms. In the third test, the total delay was 57.56536 seconds, with an average delay per second of 0.012071 seconds and a delay per millisecond of 12.07074 ms. The total delay increased significantly at maximum load compared to scenarios with fewer threads, although the average delay per second showed consistent system performance [19].

TABLE V. RESULTS OF QOS METRIC LEAST CONNECTION – DELAY CALCULATION

	1000 Thread		
Test Number-	**Total Delay's**	**Delay/s**	**Delay/ms**
1	12,99383	0,0012994	12,99383
2	10,98023	0,01098	10,98023
3	11,08898	0,011089	11,08898
	3000 Thread		
Test Number-	**Total Delay's**	**Delay/s**	**Delay/ms**
1	24,00382	0,011113	11,11288
2	36,22308	0,012304	12,30404
3	38,5215	0,013103	13,10255
	5000 Thread		
Test Number-	**test number-**	**test number-**	**test number-**
1	58,70857	0,012428	12,42773
2	58,19657	0,01234	12,34024
3	57,56536	0,012071	12,07074

Latency increased both overall and per millisecond as the number of threads was increased from 1000 to 5000. Despite the increase in the number of threads, the average latency per second remained constant, indicating that the system was able to handle the additional load thru efficient management mechanisms. Testing with 1000 threads yielded the best results due to smaller and more stable delays.

Table VI shows the results of jitter calculation for the QoS Least Connection Metric. The first test showed a delay variation of 0.002106 seconds, with jitter in seconds of 2.10811E-06 and milliseconds of 0.002108108 ms. The second test resulted in a smaller delay variation of 0.00079 seconds, with jitter of 7.90791E-07 in seconds and 0.000790791 ms in milliseconds. The third test showed a larger delay variation of 0.015244 seconds, with jitter of 1.52593E-05 in seconds and 0.015259259 ms in milliseconds. Jitter remained relatively small and stable at 1000 threads, indicating that the system is effectively managing the light load.

The first test revealed a delay variation of 0.003244 seconds, with jitter in seconds at 1.50255E-06 and milliseconds at 0.001502547 ms. The second test produced a very small delay variation of 0.000291 seconds, with jitter in seconds at 9.88787E-08 and milliseconds at 9.88787E-05 ms. The third test revealed a significantly larger delay variation of 0.988441 seconds, with jitter in seconds at 0.000336319 and milliseconds at 0.336318816 ms. At 3000 threads, jitter increased dramatically in the third test, indicating that the system was beginning to feel the strain of the increased load.

The first test with 5000 threads resulted in a delay variation of 1.070565 seconds, with jitter in seconds of 0.000226671 and in milliseconds of 0.226670548. The second test showed a smaller delay variation of 0.01748 seconds, with jitter in seconds of 3.70732E-06 and milliseconds of 0.003707317 ms. The third test revealed a significantly smaller delay variation of 0.001899 seconds, with jitter in seconds of 3.9828E-07 and milliseconds of 0.00039828 ms. Although jitter increased significantly in the first test, the results of the second and third tests indicate that the system is capable of stabilizing jitter even with a larger number of threads.

TABLE VI. RESULTS OF QOS METRIC LEAST CONNECTION - JITTER CALCULATION

Delay/s Variations	Jitter/s	Jitter/ms
	1000 Thread	
0,002106	2,10811E-06	0,002108108
0,00079	7,90791E-07	0,000790791
0,015244	1,5259E-05	0,015259259
	3000 Thread	
Delay/s Variations	**Jitter/s**	**Jitter/ms**
0,003244	1,50255E-06	0,001502547
0,000291	9,88787E-08	9,887870E-05
0,988441	0,000336319	0,336318816
	5000 Thread	
Delay/s Variations	**Jitter/s**	**Jitter/ms**
1,070565	0,000226671	0,226670548
0,01748	3,70732E-06	0,003707317
0,001899	3,9828E-07	0,00039828

Jitter increased with a higher number of threads (3000 and 5000), particularly during the initial tests. In the third test with 5000 threads, jitter stabilized again, indicating the system's ability to handle the additional load over time. With 1000 threads, the system performed optimally with minimal and consistent jitter. At 3000 threads, jitter varied, especially in the third test, suggesting that network load was beginning to affect performance. Jitter fluctuated at 3000 threads, particularly in the third test, indicating that network load was starting to impact performance. The system works very well under light to moderate loads (1000–3000 threads). For high loads (5000 threads), additional optimization is required to reduce jitter fluctuations during initial testing. The total number of packages sent and received was 1000 in all three tests (1, 2, and 3). The packet loss rate is 0%, indicating no lost packets. At 1000 threads, the Least Connection algorithm shows optimal performance, with no packet loss. This demonstrates the system's ability to handle light loads efficiently.

Table VII shows the results of the packet loss calculation. There were 1000 packets sent and received in all tests—1, 2, and 3. No packets were lost, resulting in a packet loss rate of 0%. With 1000 threads, the Least Connection algorithm performed optimally and without packet loss. This demonstrates the system's ability to handle light loads without issues. In initial testing, using 3000 threads, 3000 packets were sent; however, only 2160 were received, resulting in a loss of 840 packets. The packet loss rate was 28%, which is a significant increase. Test 2: 2944 packets were received out of 3000 sent, with 56 packets lost. The packet loss rate decreased to 1.867%. In Test 3, 2940 packets were received out of 3000 sent, with 60 packets lost. Packet loss was recorded at 2,000 percent. At 3000 threads, packet loss increased in the first test but decreased significantly in subsequent tests. This indicates that the system is beginning to be affected by the increased load while still being able to adjust its performance.

In testing involving 5000 threads, out of 5000 packets sent, 4724 packets were successfully received, resulting in 276 packets being lost. The packet loss rate was recorded at 5.52%. In the second test, 4716 packets were received out of 5000 sent, with 284 packets lost. The packet loss rate increased slightly to 5.68%. Test 3: Out of 5000 packets sent, 4769 were received, with 231 lost. The packet loss rate decreased to 4.62%. Compared to other scenarios with fewer threads, the 5000-thread scenario experienced higher packet loss. The system showed the ability to reduce packet loss in previous testing, although a large number of threads remained a challenge.

TABLE VII. PACKET LOSS CALCULATION RESULTS - SMALLEST CONNECTION QOS METRIC

Test Number	Package Shipped	Package Received	Total Packages	Package Lost
		1000 Thread		
1	1000	1000	0	0%
2	1000	1000	0	0%
3	1000	1000	0	0%
		3000 Thread		
Test Number	**Package Shipped**	**Package Received**	**Total Packages**	**Package Lost**
1	3000	2160	840	28,00%
2	3000	2944	56	1,867%
3	3000	2940	60	2,000%
		5000 Thread		
Test Number	**Package Shipped**	**Package Received**	**Total Packages**	**Package Lost**
1	5000	4724	276	5,52%
2	5000	4716	284	5,68%
3	5000	4769	231	4,62%

Packet Loss Calculation Results - Smallest Connection QoS Metric. At 1000 threads, there was no packet loss (0%), indicating optimal performance for light loads. Packet loss was high in the first test at 3000 threads (28%), but decreased significantly in the second and third tests. Packet loss increased at 5000 threads, but remained below 6%, indicating that the system was stable despite the high load. The system performed very well under light load (1000 threads). Under medium to high load (3000 and 5000 threads), the system experienced increased packet loss, but remained stable after adjustment. Under medium to high load (3000 and 5000 threads), the system experienced increased packet loss, but remained stable after adjustment. To reduce packet loss during high load (5000 threads), the network configuration must be optimized or hardware capacity increased.

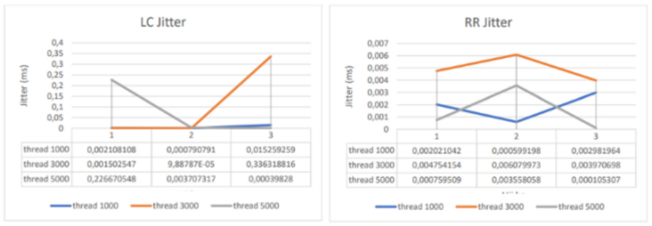
Table VIII shows the throughput results based on the Least Connection algorithm for Quality of Service (QoS) metrics. Table VIII shows the throughput results based on the Least Connection algorithm for Quality of Service (QoS) metrics. Test 2, with a total data volume of 7,813,216 bits, recorded a processing time of 10.980 seconds, resulting in a throughput of 711,5861566 bps. In Test 3, 7,813,216 bits were processed in 11.089 seconds, yielding a throughput of 704,5915772 bps. At 1000 threads, throughput remained relatively stable, averaging around 700 bps. The system maintains its high performance without significant degradation. Using 3000 threads, Test 1 processed a total of 16,876,608 bits in 24,000 seconds, resulting in a throughput of 703,074.8209 bps. The second test, with a total data of 23,020,080 bits in 36,223 seconds, yielded a throughput of 635,013.1132 bps. Test 3, with a total data of 22,971,296 bits processed in 38,522 seconds, resulted in a throughput of 596,316.2868 bps. Specifically, in the third test, throughput decreased at 3000 threads compared to 1000 threads. This indicates that system efficiency decreases as the number of threads increases.

Test 1 processed 36,910,048 bits of data in 58.709 seconds using 5000 threads, resulting in a throughput of 628.6948849 bits per second. Test 2 processed 36,847,680 bits of data in 58.197 seconds, yielding a throughput of 633.1542863 bps. Test 3, with 37,261,424 bits of data and a processing time of 57.565 seconds, achieved a throughput of 647.2930426 bits per second. Despite the increased load, throughput remained consistent at around 630 bps with 5000 threads. This demonstrates the system's ability to effectively manage heavy loads.

TABLE VIII. THROUGHPUT CALCULATION RESULTS - SMALLEST CONNECTION QOS METRIC

		1000 *Thread*	
Test Number	**Total Bit**	**Time Span(s)**	**Throughput**
1	7813312	12.994	601,3015238
2	7813216	10.980	711,5861566
3	7813216	11.089	704,5915772
		3000 *Thread*	
Test Number	**Total Bit**	**Time Span(s)**	**Throughput**
1	16876608	24.004	703,0748209
2	23002080	36.223	635,0131132
3	22971296	38.522	596,3162868
		5000 *Thread*	
Test Number	**Total Bit**	**Time Span(s)**	**Throughput**
1	36910048	58.709	628,6948849
2	36847680	58.197	633,1542863
3	37261424	57.565	647,2930426

The highest throughput was achieved at 1000 threads, with an average of over 700 bps. At 3000 threads, throughput dropped to around 600 bps, indicating the impact of increased load. At 5000 threads, throughput remained stable at approximately 630 bps, suggesting the system can handle heavy loads. The system achieved optimal throughput and faster processing times with a limited number of threads (1000). Increasing the number of threads (3000 and 5000) resulted in a slight decrease in throughput due to longer processing times. The author then compared the previously obtained calculation results with QoS comparison metrics, such as jitter, delay, throughput, and packet loss, as determined by load testing experiments. Based on the table above, the author created two graphs for each algorithm. Here are two graphs illustrating jitter in the Round Robin and Least Connection algorithms.



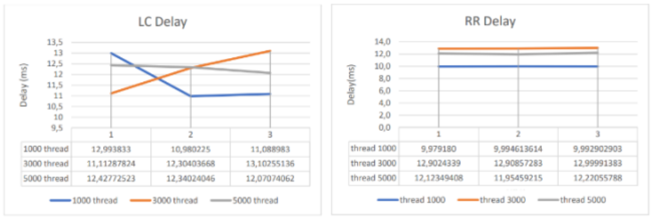
	1	2	3
thread 1000	0,002108108	0,000790791	0,015259259
thread 3000	0,001502547	9,88787E-05	0,336318816
thread 5000	0,226670548	0,003707317	0,00039828

	1	2	3
thread 1000	0,002021042	0,000599198	0,002981964
thread 3000	0,004754154	0,006079973	0,003970698
thread 5000	0,000759509	0,003558058	0,000105307

Fig. 3. Comparison graph of jitter for round robin and least connection algorithms

The graph in Figure 3 shows that the Round Robin algorithm outperforms the Least Connection algorithm in terms of jitter consistency, particularly in scenarios with heavier loads. Round Robin excels at minimizing jitter and maintaining stability across various thread load levels, making it ideal for systems requiring high data delivery time stability. Although Least Connection can adjust its performance after initial testing, it is more susceptible to jitter fluctuations when the number of threads is higher. However, it works well with a small to medium number of threads [20].

The graph above illustrates the comparison of jitter (ms) between two algorithms, Round Robin and Least Connection, for various numbers of threads (1,000, 3,000, and 5,000). Jitter is the variation in packet delivery time that affects network performance, especially for real-time applications like video streaming or VoIP. With a larger number of threads, Least Connection exhibits significant instability, as evidenced by jitter spikes, particularly in certain sessions [21]. This indicates that the algorithm performs less consistently under high loads [22]. Least Connection is more susceptible to high loads, which can lead to jitter spikes in certain sessions, especially when a large number of threads are involved. Round Robin outperforms Least Connection in terms of performance stability, with less and more consistent jitter as the number of threads increases. Round Robin is more stable and exhibits consistent jitter, even under high load. This algorithm performs better in scenarios that require low latency and consistent delivery times.



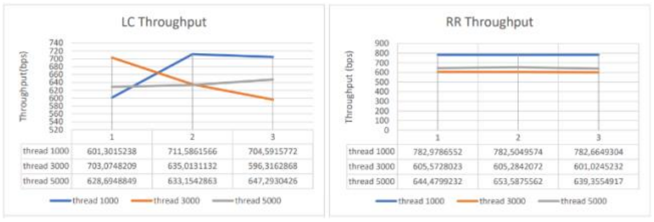
	1	2	3
1000 thread	12,993833	10,980225	11,088983
3000 thread	11,11287824	12,30403668	13,10255136
5000 thread	12,42772523	12,34024046	12,07074062

	1	2	3
thread 1000	9,979180	9,994613614	9,992902903
thread 3000	12,9024339	12,90857283	12,99991383
thread 5000	12,12349408	11,95459215	12,22055788

Fig. 4. Comparison graph of delay between round robin and least connection algorithms

The graph in Figure 4 above illustrates a comparison of delay (ms) between two load balancing algorithms, Round Robin and Least Connection, at varying numbers of threads (1,000, 3,000, and 5,000). Delay is a performance indicator in a network system that shows the time it takes to send data packets. The Least Connection algorithm exhibits variability in latency, particularly with an increasing number of threads [23], [24]. This indicates that network load can affect the performance of Least Connection, resulting in increased delay in scenarios with more threads. While Round Robin remains consistent at higher threads (3000 and 5000), Least Connection frequently experiences substantial delays.

Even as the number of threads increases, the delay performance of the Round Robin algorithm remains very consistent. For network scenarios that require time consistency, Round Robin is more reliable than Least Connection due to its stability. Even as the number of threads increases, the delay performance of the Round Robin algorithm remains very consistent. For network scenarios that require time consistency, Round Robin is more reliable than Least Connection due to its stability. Round Robin outperformed Least Connection for 1000 threads, with lower latency (around 9.97 ms). (12.99 ms during the first session). Round Robin is more suitable for applications that require consistent and stable data transmission times. The Least Connection method, while effective in certain situations, is more susceptible to high loads, potentially leading to increased latency.



LC Throughput

	1	2	3
thread 1000	601,3015238	711,5861566	704,5915772
thread 3000	703,0748209	635,0131132	596,3162868
thread 5000	628,6948849	633,1542863	647,2930426

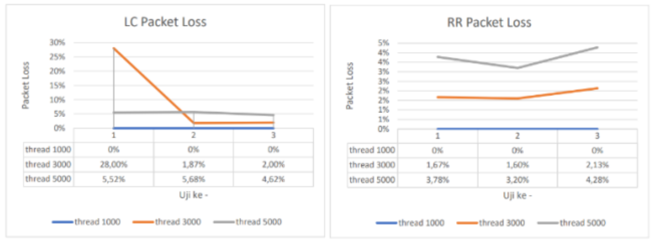
RR Throughput

	1	2	3
thread 1000	782,9786552	782,5049574	782,6649304
thread 3000	605,5728023	605,2842072	601,0245232
thread 5000	644,4799232	653,5875562	639,3554917

Fig. 5. Comparison graph of Throughput for round robin and least connection algorithms

Throughput Comparison Graph for Round Robin and Least Connection Algorithms The graph in Figure 5 illustrates the comparison of Throughput (bps) between the Round Robin (RR) and Least Connection (LC) algorithms for varying numbers of threads. (1,000, 3,000, and 5,000. Throughput refers to the amount of data successfully processed within a unit of time and is one of the network performance indicators. The throughput performance of the Least Connection algorithm varies, especially when the number of threads is increased. (3000). Decreased throughput during a specific session indicates issues with network load handling [25]. With higher loads, the fewest connections typically see a decrease in throughput. (3000 threads).

The Round Robin algorithm consistently demonstrates stable throughput performance, regardless of whether the number of threads is low or high, showing minimal fluctuations [26], [27]. Round Robin (RR) is significantly more stable than Least Connection (LC), particularly in maintaining consistent throughput values across all sessions. RR throughput is higher (782 bps) than LC (704 bps in the best session) for 1000 threads. RR maintains consistent throughput at 3000 and 5000 threads, while LC shows fluctuations that can reduce efficiency.



LC Packet Loss

	1	2	3
thread 1000	0%	0%	0%
thread 3000	28,00%	1,87%	2,00%
thread 5000	5,52%	5,68%	4,62%

RR Packet Loss

	1	2	3
thread 1000	0%	0%	0%
thread 3000	1,67%	1,60%	2,13%
thread 5000	3,78%	3,20%	4,28%

Fig. 6. Grafik perbandingan Packet Loss pada algoritma round robin dan least connection

Figure 6 compares the Packet Loss (%) of the Least Connection (LC) and Round Robin (RR) algorithms at different thread counts: 1000, 3000, and 5000. Packet loss, or the percentage of data packets lost during transmission, is an important parameter for assessing network performance. [28], [29]. The Least Connection algorithm works well with a small number of threads (1000), but packet loss increases significantly with more threads, especially in the first session. Least Connection is ideal for low-load scenarios (1000 threads) without packet loss. With more threads, the Round Robin algorithm shows relatively low and consistent packet loss, although it increases slightly in certain sessions [30]. Round Robin performs more consistently under heavy loads (3000 and 5000 threads), making it a better choice for high-load networks or scenarios requiring consistent data delivery.

IV. CONCLUSION

This study assesses the quality of service (QoS) for the Round Robin and Least Connection algorithms in load balancing using NGINX. According to the research findings, the advantages and disadvantages of both algorithms vary depending on the scenario used. The Round Robin algorithm outperforms and is consistent across almost all tested metrics, including throughput, delay, jitter, and packet loss. Round Robin distributes requests cyclically without tracking the number of connections on each server, making it a simpler and more efficient algorithm to implement. This algorithm works best with uniform requests and servers that have the same specifications. Performance decreases in scenarios involving dynamic traffic or servers with different specifications.

Least Connection is more tolerant of scenarios involving changes in server specifications and dynamic traffic. This algorithm can dynamically adjust load distribution based on the active connections of each server. Least Connection was unstable in some tests, including the first and second trials with 3000 threads, which showed anomalies in the form of significant value spikes. This means this algorithm is less suitable for scenarios with uniform servers and homogeneous traffic. Round Robin is ideal for networks with consistent demand and servers with consistent specifications. Its advantages in consistency and algorithm simplicity make it ideal for scenarios with consistent workloads. Least Connection is more suitable for complex network scenarios that include servers with varying hardware specifications or dynamic traffic. This algorithm efficiently distributes the load based on the capacity of each server.

Although Round Robin performed better in this study, the Least Connection algorithm has advantages in terms of adaptability to dynamic traffic and heterogeneous server specifications. To achieve maximum efficiency, load balancing algorithms must be tailored to the characteristics of the traffic and server infrastructure used. For future research, the authors suggest conducting testing with dynamic traffic and higher loads, as well as using various environments, including uniform and non-uniform ones.

REFERENCES

[1] O. Nathan, "Web-Based Cryptography: A Deep Dive into Data Security.," 2024.

[2] APJII, "APJII Jumlah Pengguna Internet Indonesia Tembus 221 Juta Orang," *apjii.or.id*. 2024. [Online]. Available: https://apjii.or.id/berita/d/apjii-jumlah-pengguna-internet-indonesia-tembus-221-juta-orang

[3] A. Karim, E. Ahmed, S. Azam, B. Shanmugam, and P. Ghosh, "Mitigating the Latency Induced Delay in IP Telephony Through an Enhanced De-Jitter Buffer," *Mob. Comput. Sustain. Informatics*, pp. 1–16, 2021, doi: 10.1007/978-981-16-1866-6_1.

[4] B. Veer and S. Aslam, "Distributed Computing Techniques for Real-Time Data Processing in Mobile Restaurant Systems," 2024.

[5] N. P. Dharuman, S. Avancha, V. B. R. Bhimanapati, O. Goel, N. Singh, and R. Agarwal, “Multi Controller Base Station Architecture for Efficient 2G 3G Network Operations,” *Int. J. Res. Mod. Eng. Emerg. Technol.*, vol. 12, no. 10, p. 106, 2024.

[6] D. Saxena and B. Bhowmik, “Analysis of Selected Load Balancing Algorithms in Containerized Cloud Environment for Microservices,” *VLSI Syst. Archit. Technol. Appl. (VLSI SATA)*, 2024, doi: 10.1109/vlsisata61709.2024.10560139.

[7] A. Fathima, A. Khan, M. F. Uddin, and M. M. Waris, “Optimizing Network Performance and Availability in Game Servers: A Comparative Study of Load Balancing Techniques in Sophos Firewalls,” *SSRN Electron. J.*, 2024, doi: 10.2139/ssrn.4931512.

[8] K. Meng *et al.*, “Ship Power System Network Reconfiguration Based on Swarm Exchange Particle Swarm Optimization Algorithm,” *Appl. Sci.*, vol. 14, no. 21, p. 9960, 2024, doi: 10.3390/app14219960.

[9] Z. Zhou *et al.*, “Communications with guaranteed bandwidth and low latency using frequency-referenced multiplexing,” *Nat. Electron.*, vol. 6, no. 9, pp. 694–702, 2023, doi: 10.1038/s41928-023-01022-x.

[10] Y. Kostiuk, P. Skladannyi, N. Korshun, B. Bebeshko, and K. Khorolska, “Integrated protection strategies and adaptive resource distribution for secure video streaming over a Bluetooth network,” *Inf. Technol.*, vol. 4, no. 6, pp. 14–33, 2024.

[11] P. Jaganathan and S. Balusamy, “Performance evaluation of optimization algorithms based on secured and energy conscious routing for WSN,” *AIP Conf. Proc.*, vol. 3246, p. 20004, 2024, doi: 10.1063/5.0240606.

[12] Y. Zhang, “Millimeter-Wave Wide-angle Scanning Array Antennas for 5G/6G Communication,” 2024.

[13] R. Yamada, H. Tomeba, O. Nakamura, T. Sato, and Y. Hamaguchi, “Radio Resource Allocation Using Multi-AP Coordination Considering Requirements for Video Transmission in Wireless LAN system,” *IEICE Trans. Commun.*, pp. 1–8, 2024, doi: 10.23919/transcom.2024ebt0003.

[14] L. Zhang *et al.*, “An Efficient Parallel CRC Computing Method for High Bandwidth Networks and FPGA Implementation,” *Electronics*, vol. 13, no. 22, p. 4399, 2024, doi: 10.3390/electronics13224399.

[15] M. R. Naimi, C. Zouaoui, M. Elbahri, and A. Bounoua, “Inventory of Load Balancing Parameters in MPTCP Schedulers,” *Comput. Networks*, vol. 255, p. 110880, 2024, doi: 10.1016/j.comnet.2024.110880.

[16] J. Thorpe *et al.*, “Dorylus: Affordable, scalable, and accurate ${$GNN$}$ training with distributed ${$CPU$}$ servers and serverless threads,” in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 495–514.

[17] M. Olabim, A. Greenfield, and A. Barlow, “A Differential Privacy-Based Approach for Mitigating Data Theft in Ransomware Attacks,” *Authorea (Authorea)*, 2024, doi: 10.22541/au.172625434.48862692/v1.

[18] A. Fuerst and P. Sharma, “Locality-aware Load-Balancing For Serverless Clusters,” 2022, doi: 10.1145/3502181.3531459.

[19] J. Logeshwaran, N. Shanmugasundaram, and J. Lloret, “L-RUBI: An efficient load-based resource utilization algorithm for bi-partite scatternet in wireless personal area networks,” *Int. J. Commun. Syst.*, vol. 36, no. 6, 2023, doi: 10.1002/dac.5439.

[20] C. Stylianopoulos *et al.*, “On the performance of commodity hardware for low latency and low jitter packet processing,” *Chalmers Res. (Chalmers Univ. Technol.*, 2020, doi: 10.1145/3401025.3403591.

[21] Y. Chen *et al.*, “TDSQL: Tencent Distributed Database System,” *Distrib. Database Syst. PVLDB*, vol. 17, no. 12, pp. 3869–3882, 2024, doi: 10.14778/3685800.3685812.

[22] A. Seyfollahi and A. Ghaffari, “A lightweight load balancing and route minimizing solution for routing protocol for low-power and lossy networks,” *Comput. Networks*, vol. 179, p. 107368, 2020, doi: 10.1016/j.comnet.2020.107368.

[23] R. Mishra, I. Ahmad, and A. Sharma, “An energy-efficient queuing mechanism for latency reduction in multi-threading,” *Sustain. Comput. Informatics Syst.*, vol. 30, p. 100462, 2021, doi: 10.1016/j.suscom.2020.100462.

[24] D. L. Freire, R. Z. Frantz, and F. R. Frantz, “Towards optimal thread pool configuration for run-time systems of integration platforms,” *Int. J. Comput. Appl. Technol.*, vol. 62, no. 2, p. 129, 2020, doi: 10.1504/ijcat.2020.104692.

[25] L. F. Eliyan and R. Di Pietro, “DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges,” *Futur. Gener. Comput. Syst.*, vol. 122, 2021, doi: 10.1016/j.future.2021.03.011.

[26] M. Prajana, H. Rajkumar, A. S. Sannidhi, K. Vidyasri, and S. Krishnan, “Adaptive Multi-Level Feedback Round-Robin,” *Int. Conf. Comput. Commun. Netw. Technol.*, pp. 1–9, 2024, doi: 10.1109/icccnt61001.2024.10724187.

[27] A. Johansson, “HTTP Load Balancing Performance Evaluation of HAProxy, NGINX, Traefik and Envoy with the Round-Robin Algorithm,” *DIVA*. 2022. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1678660

[28] S. Suryadevara, “Real-Time Task Scheduling Optimization in WirelessHART Networks: Challenges and Solutions,” *Int. J. Adv. Eng. Technol. Innov.*, vol. 1, no. 3, pp. 29–55, 2022.

[29] C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, “Performance Evaluation of Blockchain Systems: A Systematic Survey,” *IEEE Access*, vol. 8, no. 1, pp. 126927–126950, 2020, doi: 10.1109/access.2020.3006078.

[30] A. K. Chaurasia *et al.*, “Simmer: Rate proportional scheduling to reduce packet drops in vGPU based NF chains,” vol. 197, no. 44, pp. 1–11, 2022, doi: 10.1145/3545008.3545068.