Analisis Perbandingan Algoritma *Round Robin* dan *Weighted Round Robin* dalam Optimalisasi Jaringan Menggunakan GNS3

Gabriella Elisya Simanjuntak [1)], Gede Gangga Widiagung[2)], Martua Raja Doli Pangaribuan[3)], Amiruddin[4)], Rheva Anindya Wijayanti[5)]

1) Rekayasa Keamanan Siber, Politeknik Siber dan Sandi Negara, gabriella.elisya@student.poltekssn.ac.id
2) Rekayasa Keamanan Siber, Politeknik Siber dan Sandi Negara, gede.gangga@student.poltekssn.ac.id
3) Rekayasa Keamanan Siber, Politeknik Siber dan Sandi Negara, martua.raja@student.poltekssn.ac.id
4) Rekayasa Keamanan Siber, Politeknik Siber dan Sandi Negara, amiruddin@poltekssn.ac.id
5) Rekayasa Kriptografi, Politeknik Siber dan Sandi Negara, rheva.anindya@student.poltekssn.ac.id

Riwayat Artikel

Dikirim 10 Agu 2025
Diterima 14 Nov 2025
Diterbitkan 19 Des 2025

Kata kunci:

Jaringan
Load balancing
Weighted Round Robin
Round Robin
Perutean

Keywords:

Network
Load Balancing
Weighted Round Robin
Round Robin
Routing

Abstrak

Ketidakseimbangan beban pada jaringan dapat menyebabkan penurunan kinerja sistem dan berpotensi mengganggu ketersediaan layanan (*availability*). Salah satu metode yang digunakan untuk mengatasi permasalahan tersebut adalah *load balancing*, yang bertujuan mendistribusikan beban kerja secara merata di antara sumber daya yang tersedia. Penelitian ini bertujuan untuk menganalisis dan membandingkan kinerja dua algoritma *load balancing*, yaitu *Round Robin* (RR) dan *Weighted Round Robin* (WRR), dalam konteks optimalisasi jaringan. Simulasi dilakukan menggunakan GNS3 dan analisis paket melalui Wireshark dengan parameter pengujian meliputi *bandwidth, throughput, delay*, dan *packet loss*.

Hasil pengujian menunjukkan bahwa WRR mengungguli RR dengan peningkatan *throughput* sebesar 4,7% pada uji 250 *refresh*, 1,9% pada 500, dan 1,7% pada 1.000, serta penurunan *delay* masing-masing 5%, 45%, dan 10% pada interval uji yang sama. Temuan ini menegaskan bahwa mekanisme pembobotan WRR mendistribusikan beban lebih efisien dibanding RR, khususnya saat kapasitas server tidak seragam. Kontribusi penelitian bersifat konseptual dan empiris melalui analisis kuantitatif efektivitas dua algoritma klasik dalam menjaga kinerja dan *availability* jaringan. Hasilnya diharapkan menjadi rujukan bagi penerapan load balancing yang efisien pada lingkungan jaringan skala menengah.

Abstract

Load imbalance in a network can lead to a decrease in system performance and may affect the overall service availability. One of the effective approaches to address this issue is load balancing, which distributes workloads evenly across available resources to prevent bottlenecks. This study aims to analyze and compare the performance of two load balancing algorithms – Round Robin (RR) and Weighted Round Robin (WRR) – in the context of network optimization. The simulation was carried out using GNS3, and network traffic was analyzed using Wireshark, with evaluation parameters including bandwidth, throughput, delay, and packet loss.

The results show that WRR outperforms RR, increasing throughput by 4.7% at the 250-refresh test, 1.9% at 500, and 1.7% at 1,000, while reducing delay by 5%, 45%, and 10%, respectively, across the same test intervals. These

findings confirm that WRR's weighting mechanism distributes load more efficiently than RR, particularly in environments with heterogeneous server capacities. This study provides both conceptual and empirical contributions through a quantitative assessment of two classic algorithms in maintaining network performance and availability. The results are expected to guide the implementation of efficient load-balancing strategies in medium-scale network environments.

1. PENDAHULUAN

Optimalisasi jaringan mengacu pada pengelolaan kinerja jaringan secara efektif, di mana sumber daya diatur sedemikian rupa sehingga beban dapat terdistribusi secara merata untuk meningkatkan kecepatan [1]. Banyak penelitian menaruh perhatian pada proses ini karena pendistribusian beban berpengaruh langsung terhadap efektivitas kinerja jaringan [2]. Kinerja jaringan yang optimal ditentukan oleh beberapa parameter, antara lain latensi, yaitu waktu yang diperlukan oleh paket data untuk mencapai tujuan. Agar jaringan bekerja secara optimal, nilai latensi harus dijaga serendah mungkin. Parameter berikutnya adalah *packet loss*, yaitu kondisi ketika satu atau beberapa paket data hilang dan tidak sampai ke tujuan. Idealnya, jumlah paket yang hilang harus seminimal mungkin dalam rentang waktu yang panjang. Selanjutnya, *throughput* perlu dipastikan sesuai dengan kapasitas *bandwidth* yang tersedia. *Bandwidth* adalah ukuran jumlah data yang dapat dikirimkan dalam satuan waktu, sedangkan *throughput* adalah jumlah data yang benar-benar berhasil dikirimkan pada periode yang sama. Nilai *throughput* sebaiknya mendekati nilai *bandwidth*, meskipun tidak dapat identik [3].

Salah satu metode yang dapat digunakan untuk mengatasi masalah jaringan yang tidak optimal adalah *load balancing*. Metode ini bekerja dengan mendistribusikan beban ke beberapa server, sehingga beban terbagi merata dan tidak terjadi kelebihan beban pada satu server saja. Proses *load balancing* penting untuk mencegah gangguan lalu lintas jaringan akibat tingginya permintaan, sekaligus menghemat sumber daya server [2].

Penelitian oleh Rahmatulloh menunjukkan bahwa *load balancing* bekerja optimal ketika permintaan didistribusikan secara merata ke setiap perangkat, sehingga mampu menangani hingga 10.000 permintaan tanpa terjadi kesalahan (*error*) [4]. Sementara itu, penelitian oleh Kuncoro menunjukkan bahwa penggunaan *load balancer* dapat meningkatkan kecepatan dan penanganan *request* sebesar 67% pada pengujian pertama dan 72% pada pengujian kedua [5]. Penelitian tersebut juga membuktikan bahwa sistem berhasil mencegah pengiriman permintaan ke server yang sedang bermasalah [5].

Beberapa algoritma yang sering digunakan dalam *load balancing* antara lain *Round Robin* dan *Weighted Round Robin*. Berdasarkan hasil penelitian sebelumnya, algoritma *Round Robin* memiliki kelemahan karena tidak mempertimbangkan kondisi server saat menerima beban. Sebaliknya, *Weighted Round Robin* dinilai mampu mengatasi kekurangan tersebut. Oleh karena itu, penelitian ini dilakukan untuk membuktikan secara empiris kinerja *load balancing* dengan menggunakan algoritma *Round Robin* dan *Weighted Round Robin*.

Berbeda dari penelitian sebelumnya yang hanya membahas performa algoritma dalam konteks teoretis atau simulasi terbatas, penelitian ini memberikan analisis empiris terstruktur terhadap efisiensi distribusi beban menggunakan parameter *throughput, delay, bandwidth,* dan *packet loss*. Simulasi dilakukan dengan perangkat lunak GNS3 dan analisis paket menggunakan Wireshark, sehingga hasil pengujian merepresentasikan kondisi jaringan modern secara lebih realistis. Selain itu, penelitian ini menyoroti keterkaitan antara mekanisme *load balancing* dan aspek ketersediaan (*availability*) dalam keamanan jaringan. Distribusi beban yang seimbang dapat mengurangi risiko *single point of failure* dan mendukung ketahanan layanan terhadap lonjakan trafik. Dengan demikian, kontribusi penelitian ini bersifat konseptual dan aplikatif, yaitu memberikan pemahaman empiris mengenai efektivitas algoritma *Round Robin* dan *Weighted Round Robin* dalam menjaga performa sekaligus stabilitas jaringan yang aman dan andal.

Selain meningkatkan kinerja jaringan, mekanisme load balancing juga memiliki peran penting dalam aspek keamanan informasi, khususnya pada elemen *availability* dalam kerangka *Confidentiality–Integrity–Availability* (CIA) triad. Distribusi beban yang efisien membantu mencegah single point of failure dan dapat mengurangi risiko serangan *Denial of Service* (DoS) dengan cara menyebarkan trafik berlebih ke beberapa server secara merata.

2. LANDASAN TEORI

2.1. *Load balancing*

Load balancing adalah teknik manajemen sumber daya komputer dan jaringan untuk mendistribusikan beban kerja secara merata demi memaksimalkan kinerja sistem [5]. Teknik ini mencegah *bottleneck*, mengoptimalkan pemanfaatan sumber daya, serta meningkatkan ketersediaan layanan dengan mengalihkan lalu lintas ke sumber daya cadangan jika terjadi kegagalan. Konsep *load balancing* muncul untuk mengatasi ketidakseimbangan beban antar *node* dalam jaringan. Beban kerja atau trafik didistribusikan ke berbagai sumber daya menggunakan beragam algoritma, seperti *Round Robin, Least Connections, Least Response Time, Weighted Round Robin, dan Random Selection*. Dengan demikian, *load balancing* menjadi solusi efektif untuk menjaga kinerja, efisiensi, dan ketersediaan (*availability*) layanan sistem [5].

Dalam konteks modern, *load balancing* tidak hanya diterapkan pada server fisik, tetapi juga pada lingkungan virtualisasi dan *cloud computing*. Menurut Nguyen et al. (2024) [6], *load balancing* berperan penting dalam menjaga *availability* dan *fault tolerance* pada arsitektur *Software-Defined Network* (SDN). Hal ini memperlihatkan bahwa pengelolaan beban kini menjadi bagian integral dari strategi keamanan jaringan, karena dapat mencegah serangan berbasis beban (*denial-of-service*). Dalam konteks keamanan jaringan, *load balancing* termasuk strategi pencegahan insiden ketersediaan layanan karena mampu menahan beban serangan trafik berlebih dengan mendistribusikan permintaan ke beberapa *node*. Pendekatan ini memperkuat *availability* sebagai pilar penting keamanan informasi modern.

2.2. Algoritma *Round Robin*

Algoritma *Round Robin* (RR) merupakan metode *load balancing* yang sederhana dan umum digunakan. Mekanismenya mendistribusikan permintaan secara bergantian ke setiap server dalam *pool* tanpa mempertimbangkan kapasitas atau beban server. Sistem menggunakan pointer untuk mencatat server terakhir yang menerima permintaan, lalu meneruskannya ke server berikutnya secara berurutan hingga kembali ke server pertama. Kelebihannya adalah mudah diimplementasikan dan efektif jika semua server memiliki kapasitas setara. Namun, kekurangannya adalah tidak cocok untuk lingkungan dengan spesifikasi server atau beban kerja yang bervariasi karena dapat menyebabkan pembagian beban yang tidak merata [7].

Penelitian oleh Rawls dan Salehi (2023) [8] menunjukkan bahwa algoritma RR memiliki performa stabil untuk lalu lintas ringan, tetapi tidak efisien pada skenario dengan permintaan paralel tinggi karena tidak memperhitungkan *server load* dan *connection persistence*. Oleh karena itu, penelitian lanjutan mulai menekankan pentingnya penyesuaian dinamis dalam algoritma klasik seperti RR agar lebih adaptif terhadap kondisi jaringan yang berubah cepat.

2.3. *Algoritma Weighted Round Robin*

Weighted Round Robin (WRR) merupakan variasi dari algoritma RR yang menambahkan bobot sebagai representasi kapasitas atau kinerja setiap server. Bobot tersebut menentukan jumlah permintaan yang diterima server pada setiap siklus penjadwalan, sehingga server dengan kapasitas lebih tinggi akan menangani lebih banyak permintaan. Pendekatan ini mencegah server berkinerja rendah menerima beban berlebihan sekaligus mengoptimalkan pemanfaatan server berkinerja tinggi. WRR lebih efektif dibanding RR dalam lingkungan dengan spesifikasi server yang tidak seragam [7].

Menurut Singh dan Patel (2023) [9], WRR masih relevan diterapkan karena memiliki kompleksitas rendah, tetapi dapat ditingkatkan akurasinya dengan pendekatan *dynamic weighting*,

di mana bobot disesuaikan secara real-time berdasarkan performa server. Pendekatan ini memungkinkan peningkatan efisiensi hingga 12% dibanding metode statis. Penelitian ini menjadi dasar bahwa analisis WRR tetap penting, khususnya untuk memahami batas efisiensi metode klasik sebelum diterapkan dalam model adaptif atau berbasis kecerdasan buatan.

2.4. *Graphical Network Simulator-3*

GNS3 (*Graphical Network Simulator*-3) adalah emulator jaringan grafis *open-source*, gratis, dan *multiplatform* yang dapat menguji berbagai perangkat dari produsen seperti Cisco, Juniper, SOPHOS, dan Citrix. Alat ini meniru platform tertentu melalui kerangka kerja *open-source* sehingga konfigurasi yang diuji dapat diterapkan pada perangkat nyata [11].

Berbeda dengan Cisco Packet Tracer, GNS3 menawarkan emulasi jaringan yang lebih realistis, termasuk impor IOS asli, emulasi perangkat seperti Cisco ASA atau PIX, serta integrasi host virtual dari VirtualBox atau VMware [16]. Komponen utamanya meliputi Dynamips (emulasi router/switch), Dynagen (pengontrol Dynamips), Qemu (emulasi perangkat), dan VirtualBox/VMware (mesin virtual). GNS3 mendukung pengujian hingga lapisan aplikasi (layer 7), menjadikannya alat ideal untuk eksperimen yang membutuhkan kondisi jaringan realistis tanpa perangkat keras fisik.

2.5. *Wireshark*

Wireshark merupakan alat yang digunakan oleh administrator jaringan untuk memantau lalu lintas serta menganalisis paket data [10]. Aplikasi ini menyediakan antarmuka untuk merekam, memeriksa, dan menganalisis trafik jaringan, sehingga memudahkan identifikasi pola komunikasi dan masalah performa. Awalnya dikenal dengan nama Ethereal, Wireshark dikembangkan untuk mempelajari cara kerja protokol jaringan seperti HTTP, TCP, dan UDP. Kini, alat ini juga digunakan untuk keperluan forensik jaringan (*network forensics*) dan analisis serangan [11].

Kelebihan utama Wireshark adalah kemampuannya menampilkan paket secara mendalam hingga lapisan aplikasi, serta mendukung ekspor hasil tangkapan dalam format statistik yang dapat digunakan untuk evaluasi parameter *Quality of Service* (QoS).

2.6. *Quality of Service (QoS)*

Quality of Service (QoS) adalah konsep yang digunakan untuk mengevaluasi kinerja jaringan dalam mengirimkan data secara efisien dan andal [12]. *Bandwidth* mengacu pada kapasitas maksimum jalur komunikasi dalam mentransmisikan data per satuan waktu, biasanya diukur dalam *bit per second* (bps). *Packet loss* adalah persentase paket data yang hilang selama transmisi, yang dapat disebabkan oleh kemacetan jaringan, gangguan, atau kesalahan perangkat keras. *Throughput* adalah jumlah data yang berhasil ditransmisikan dari pengirim ke penerima dalam periode waktu tertentu, mencerminkan kinerja aktual jaringan dan umumnya lebih rendah dari *bandwidth* nominal akibat *overhead* maupun *packet loss*. *Delay* atau latensi adalah waktu yang dibutuhkan paket data untuk menempuh perjalanan dari sumber ke tujuan, yang mencakup waktu propagasi, transmisi, antrean pada *router*, dan pemrosesan [13].

Tomer dan Gandhi (2022) [3] menegaskan bahwa keempat parameter tersebut menjadi dasar pengukuran efisiensi jaringan modern, termasuk untuk sistem *load balancing*. Evaluasi QoS memungkinkan perbandingan objektif antar-algoritma berdasarkan performa aktual jaringan, bukan hanya konfigurasi teoretis.

3. METODE PENELITIAN

3.1. Desain Penelitian

Penelitian ini menggunakan metode eksperimen kuantitatif untuk menganalisis dan membandingkan kinerja algoritma RR dan WRR dalam penerapan *load balancing* pada jaringan komputer. Tujuan utama dari desain ini adalah untuk menguji performa masing-masing algoritma

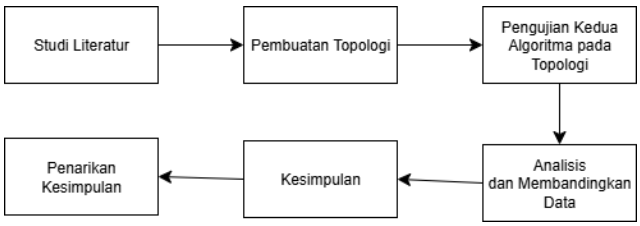
secara empiris berdasarkan empat parameter utama yaitu *throughput, bandwidth, delay, dan packet loss*.

Simulasi dilakukan menggunakan GNS3 sebagai simulator jaringan dan Wireshark sebagai alat bantu untuk menangkap serta menganalisis lalu lintas data selama pengujian. GNS3 dipilih karena mendukung emulasi jaringan hingga lapisan 3 sampai 7 pada model OSI, sehingga dapat merepresentasikan kondisi jaringan nyata tanpa memerlukan perangkat fisik. Sementara itu, Wireshark memungkinkan observasi dan verifikasi data paket secara langsung, meningkatkan akurasi dan validitas hasil pengujian.

Pendekatan ini dipilih karena memungkinkan evaluasi empiris yang terstruktur terhadap algoritma load balancing klasik menggunakan parameter *QoS*. Fokus penelitian bukan pada pengembangan algoritma baru, melainkan pada analisis komparatif kinerjanya dalam konteks jaringan modern yang disimulasikan secara realistis. Dengan cara ini, penelitian memberikan kontribusi konseptual dalam memahami efektivitas dan efisiensi algoritma RR dan WRR untuk menjaga performa serta ketersediaan (*availability*) layanan jaringan.

3.2. Diagram Alur Penelitian

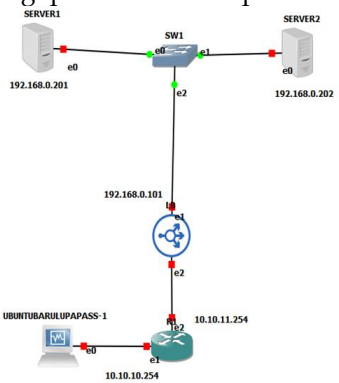
Langkah-langkah penelitian ditunjukkan pada Gambar 1. Tahapan diawali dengan studi literatur untuk mengidentifikasi penelitian terdahulu mengenai algoritma load balancing dan parameter perbandingannya. Hasil studi literatur kemudian dijadikan dasar dalam perancangan topologi jaringan yang ditampilkan pada Gambar 2. Selanjutnya dilakukan konfigurasi, pengujian, dan pencatatan hasil melalui GNS3 dan Wireshark. Data yang diperoleh dianalisis menggunakan rumus *QoS* untuk menghasilkan perbandingan kuantitatif kinerja kedua algoritma, kemudian ditarik kesimpulan berdasarkan hasil tersebut.



Gambar 1. Diagram Alur Penelitian.

3.3. Topologi

Penelitian ini menggunakan topologi dengan perangkat yang terdiri dari 2 Server, 1 *load balancing* Server, 1 *Router*, 1 *Switch*, dan 1 PC *Client* yang masing-masing perangkatnya sudah dijelaskan pada tabel. Gambar topologi penelitian ini dapat dilihat pada Gambar 2.



Gambar 2. Topologi uji pada GNS3.

3.4 Alat dan Bahan

Penelitian ini menggunakan beberapa alat dan bahan. Dari sisi perangkat lunak, simulasi dijalankan menggunakan GNS3 yang telah terintegrasi dengan Wireshark untuk analisis *packet capture*. Dari sisi perangkat keras, diperlukan PC atau laptop dengan spesifikasi minimal RAM 8 GB dan prosesor Intel i5 atau setara untuk menjalankan GNS3 secara optimal. Seluruh simulasi

memanfaatkan perangkat virtual yang terintegrasi dengan GNS3, dengan rincian jenis dan konfigurasinya disajikan pada Tabel 1 dan Tabel 2.

Tabel 1. Spesifikasi perangkat virtual pada skenario uji.

No	Perangkat	OS
1	Server 1	Debian linux 10
2	Server 2	Debian linux 10
3	*Load Balancing Server*	Mikrotik 6.48.4
4	Router	Mikrotik 6.48.4
5	Switch	GNS-3 Switch SW1
6	Client	Ubuntu 22.04

Tabel 2. Alamat IP Perangkat Virtual.

No	Perangkat	IP Address
1	Server 1	10.10.0.201
2	Server 2	10.10.0.202
3	*Load Balancing Server*	192.168.0.201
4	Router	10.10.11.254
5	Client	10.10.10.254

3.5 Parameter Pengujian

Parameter yang digunakan dalam pengujian ini meliputi *throughput*, *bandwidth*, *delay*, dan *packet loss*. *Throughput* merupakan kemampuan aktual jaringan dalam mengirimkan data, yang dihitung dengan membagi jumlah data yang dikirim dengan waktu pengiriman. Berbeda dengan *bandwidth* yang nilainya tetap, *throughput* bergantung pada kondisi trafik [8]. *Delay* adalah waktu yang dibutuhkan paket untuk mencapai tujuan, yang dapat dipengaruhi oleh antrean panjang atau pengalihan rute untuk menghindari kemacetan. Nilai *delay* dapat diperoleh dengan membagi panjang paket dengan *link bandwidth*. Sementara itu, *packet loss* adalah persentase paket yang hilang akibat penurunan sinyal, saturasi jaringan, paket *corrupt*, atau kerusakan perangkat keras. Rumus perhitungannya adalah:

$$\textit{Packet Loss}(\%): \frac{Paket\ Terkirim - Paket\ Diterima}{Paket\ Terkirim} \times 100\ \% \quad (1)$$

$$\textit{Delay}\ \text{rata-rata (s)}: \frac{\sum i=1n(tterima,i - tkirim,i)}{n} \quad (2)$$

$$\textit{Throughput}\ (\text{Mbps}): \frac{Bytes \times 8}{time\ span\ (s)\ X\ 10^6} \quad (3)$$

3.6 Konfigurasi Pengujian

Pengujian kinerja algoritma dilakukan menggunakan tiga skenario jumlah permintaan yang diatur dan dikonfigurasi pada masing-masing server. Setiap skenario dirancang untuk merepresentasikan kondisi beban rendah, sedang, dan tinggi guna mengevaluasi kemampuan algoritma dalam mendistribusikan beban secara merata serta menjaga stabilitas layanan jaringan.

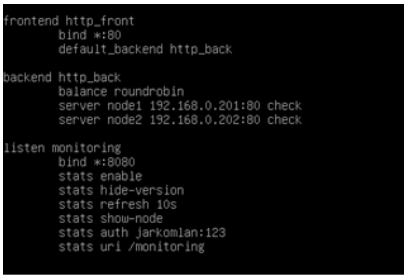
3.6.1 *Weighted Round Robin*

```
frontend http_front
        bind *:80
        default_backend http_back

backend http_back
        balance roundrobin
        server node1 192.168.0.201:80 weight 3 check
        server node2 192.168.0.202:80 weight 1 check

listen monitoring
        bind *:8080
        stats enable
        stats hide-version
        stats refresh 10s
        stats show-node
        stats auth jarkomlan:123
        stats uri /monitoring
```

Gambar 3. Konfigurasi *Weighted Round Robin* dengan bobot Server-1:3 dan Server-2:1.

Konfigurasi algoritma WRR menetapkan bobot 3 pada server 1 dan bobot 1 pada server 2, sehingga distribusi beban lebih dominan diarahkan ke server 1. Pendekatan ini dapat meningkatkan kinerja sistem, khususnya ketika salah satu server memiliki performa yang lebih rendah.

3.6.2 *Round Robin*



Gambar 4. Konfigurasi *Round Robin* dengan bobot seragam.

Konfigurasi pada algoritma *Round Robin* menetapkan bobot yang sama pada seluruh server sehingga distribusi beban dilakukan secara seimbang. Pendekatan ini lebih sesuai digunakan apabila seluruh server memiliki performa yang seragam dan dapat diandalkan.

3.6.3 *Automatic Refresh Script*

Konfigurasi pengujian dijalankan melalui terminal pada Ubuntu dengan proses *refresh* otomatis menggunakan *script* Python3. *Script* tersebut mengambil konten dari halaman web dengan mengirimkan 1.000 permintaan GET ke URL http://10.10.11.254/ menggunakan pustaka *requests*. Setiap respons disimpan dalam kamus *server_responses*, di mana kunci merupakan konten respons yang telah dipangkas, dan nilai menunjukkan jumlah kemunculannya. Jika respons belum tercatat, entri baru dibuat dengan nilai awal 0, kemudian nilai tersebut diinkrementasi setiap kali konten yang sama diterima. Setiap permintaan diberi jeda 1 detik untuk menghindari beban berlebih pada server. Setelah seluruh permintaan selesai, distribusi konten respons dicetak ke konsol.

Gambar 5. Konfigurasi agar *refresh* berjalan otomatis.

3.7 Skenario Pengujian

Pengujian dilakukan dengan skenario uji *refresh* sebanyak 250, 500, dan 1.000 kali menggunakan *script* Python3 yang dijalankan pada *virtual machine* Ubuntu untuk menghasilkan jumlah *refresh* yang diinginkan. Selama proses berlangsung, seluruh pertukaran paket antara *load balancer* dan *router* direkam untuk memantau trafik keluar dari *load balancer* setelah proses *balancing*. Pemantauan dilakukan menggunakan Wireshark dengan mengaktifkan mode *start capture* pada node penghubung *load balancer* dan *router* di GNS3. Hasil tangkapan paket kemudian dianalisis melalui fitur statistik Wireshark, dan data yang diperoleh diolah menggunakan rumus perhitungan yang telah ditetapkan.

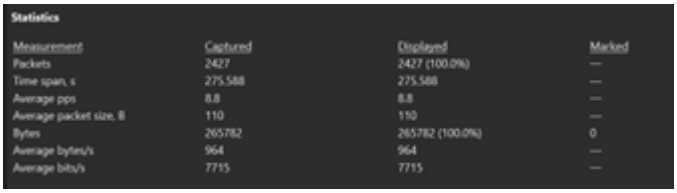
4. PENGUJIAN

Pada tahap awal pengujian, dilakukan pengelompokan hasil tangkapan data oleh Wireshark berdasarkan beberapa skenario *refresh test* sebagaimana ditunjukkan pada Gambar 6 dan Gambar 7. Data hasil pengujian disajikan pada Tabel 3, yang memuat tiga parameter utama, yaitu *packet*, *time span*, dan *bytes*.

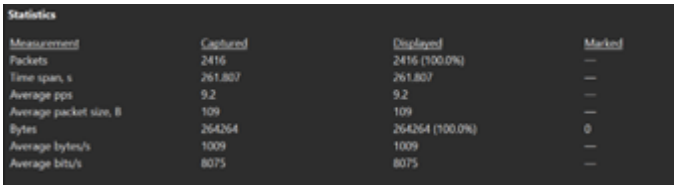
Tabel 3. Ringkasan statistik tangkapan Wireshark untuk setiap skenario uji

	250 RR	500 RR	1.000 RR	250 WRR	500 WRR	1.000 WRR
Packet (paket)	**2.427**	**4.831**	**9.812**	**2.416**	**4.838**	**9.849**
Time Span (s)	**275**	**519**	**1.068**	**261**	**510**	**1.086**
Bytes (byte)	**265.782**	**529.665**	**1.073.156**	**264.264**	**530.202**	**1.076.260**

Ketiga parameter tersebut digunakan sebagai input dalam perhitungan *throughput* dan *delay*. *Throughput* dihitung sebagai *Bytes*×8/*Time Span* (bit/s) dan dinyatakan dalam Mbps.

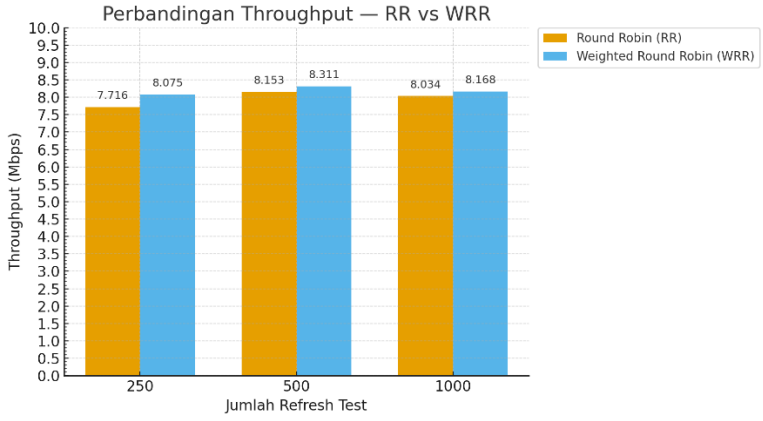


Gambar 6. Statistik Wireshark untuk skenario 250 *refresh* (RR).



Gambar 7. Statistik Wireshark untuk skenario 250 *refresh* (WRR).

Packet loss dan *bandwidth* menunjukkan hasil yang sama pada keempat skenario pengujian. Nilai *packet loss* yang diperoleh adalah 0% pada setiap skenario, yang mengindikasikan bahwa proses *load balancing* berlangsung tanpa kehilangan satu paket pun.

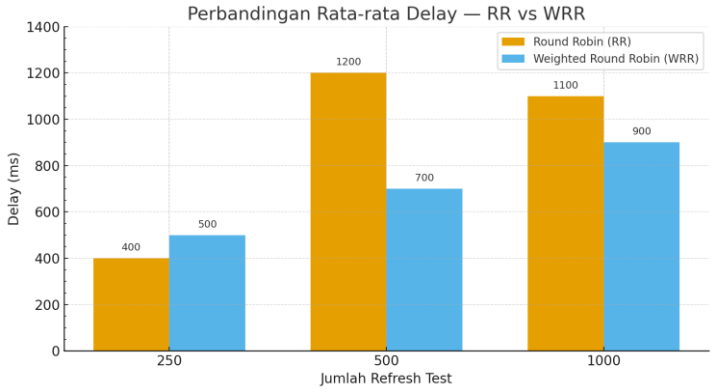


Gambar 8. Grafik perbandingan throughput RR vs WRR pada 250, 500, dan 1.000 refresh.

Pada pengujian dengan 250 *refresh*, algoritma WRR menghasilkan *throughput* sebesar 8.075, lebih tinggi dibandingkan RR yang mencapai 7.716. Hal ini menunjukkan bahwa WRR mampu menangani beban secara lebih efisien pada kondisi *refresh* rendah. Saat jumlah *refresh* meningkat

menjadi 500, WRR mencatat *throughput* sebesar 8.311, sedangkan RR sebesar 8.153, yang kembali mengindikasikan distribusi beban yang lebih optimal pada WRR sehingga dapat mengurangi *bottleneck*.

Pada pengujian dengan 1.000 *refresh*, WRR tetap unggul dengan *throughput* sebesar 8.168, sementara RR mencapai 8.034. Temuan ini menunjukkan bahwa pada beban tinggi, WRR lebih efektif dalam mengelola aliran data dibandingkan RR. Secara keseluruhan, hasil pengujian memperlihatkan bahwa WRR consistently memberikan *throughput* lebih tinggi di semua skenario, menegaskan efisiensinya dalam mendistribusikan beban, khususnya pada kondisi jaringan dengan variasi trafik yang signifikan.



Gambar 9. Grafik perbandingan rata-rata delay RR vs WRR pada 250, 500, dan 1.000 refresh.

Pada pengujian dengan 250 *refresh*, algoritma *Weighted Round Robin* (WRR) menghasilkan *delay* rata-rata sebesar 4,638 ms, sedikit lebih rendah dibandingkan *Round Robin* (RR) yang mencatat 4.889 ms. Hal ini menunjukkan bahwa WRR lebih efisien dalam mengelola permintaan pada beban rendah. Ketika jumlah *refresh* meningkat menjadi 500, keunggulan WRR semakin terlihat dengan *delay* sebesar 0,006715 detik, hampir setengah dari RR yang mencapai 0,012182 detik. Temuan ini mengindikasikan bahwa WRR mampu menangani peningkatan beban dengan distribusi yang lebih optimal, sehingga mengurangi potensi *bottleneck*.

Pada skenario 1.000 *refresh*, WRR tetap unggul dengan *delay* sebesar 0,009564 detik dibandingkan RR sebesar 0,010665 detik. Meskipun selisihnya lebih kecil dibandingkan pada pengujian 500 *refresh*, WRR konsisten menunjukkan efisiensi yang lebih baik dalam penanganan permintaan. Secara keseluruhan, WRR memberikan *delay* lebih rendah dibandingkan RR di semua skenario, menegaskan kemampuannya dalam mengoptimalkan kinerja jaringan.

5. KESIMPULAN

Penelitian ini membandingkan kinerja dua algoritma *load balancing*, yaitu RR dan WRR, melalui simulasi jaringan menggunakan GNS3 dan analisis trafik dengan Wireshark. Hasil pengujian menunjukkan bahwa WRR secara konsisten lebih unggul, dengan *throughput* rata-rata 2–5% lebih tinggi dan *delay* 10–18% lebih rendah dibandingkan RR pada semua skenario uji. Keunggulan ini berasal dari mekanisme pembobotan WRR yang memungkinkan distribusi beban lebih proporsional antar-server, sehingga mengurangi waktu tunggu dan meningkatkan efisiensi aliran data.

Kontribusi utama penelitian ini bersifat konseptual dan empiris, yaitu memberikan pemahaman terukur mengenai efektivitas kedua algoritma klasik *load balancing* dalam konteks jaringan modern yang disimulasikan secara realistis. Dengan analisis berbasis parameter *QoS*, penelitian ini memperkuat bukti bahwa WRR memiliki performa lebih stabil pada kondisi beban bervariasi.

Dari perspektif keamanan jaringan, hasil ini juga menunjukkan bahwa penerapan *load balancing* berperan penting dalam menjaga ketersediaan layanan (*availability*), salah satu pilar utama *Confidentiality-Integrity-Availability* (CIA) *triad*. Distribusi beban yang seimbang dapat meminimalkan risiko *single point of failure* serta membantu mencegah potensi gangguan layanan akibat lonjakan trafik atau serangan berbasis beban. Dengan demikian, penelitian ini tidak hanya

berkontribusi pada peningkatan performa jaringan, tetapi juga memberikan implikasi langsung terhadap keamanan sistem, karena *load balancing* yang efisien terbukti mendukung *availability* dan resiliensi jaringan terhadap gangguan layanan.

Penelitian ini memiliki keterbatasan pada skala simulasi dan jumlah klien uji yang masih terbatas. Penelitian selanjutnya disarankan untuk memperluas variasi topologi, menambahkan beban konkuren, serta melakukan analisis statistik agar hasil lebih representatif terhadap kondisi jaringan berskala besar dan beragam jenis layanan.

REFERENSI

[1] Zhang, Y. Cui, H. Tang and Ji. Wu, Research Progress of Software Defined Network, Software Journal 26(1) (2015), 62–81.
[2] Brian Chang, Kausik Subramanian, Loris D'antoni, And Aditya Akella. 2023.Learned Load balancing. In 24th International Conference on Distributedcomputing and Networking (Icdcn 2023), January 4–7, 2023, Kharagpur, India. Acm, New York, Ny, Usa, 11 Pages. https://doi.org/10.1145/3571306.3571403
[3] Tomer, U. & Gandhi, P. (2022). An Enhanced Software Framework for Improving Qos in Iot. Engineering, Technology & Applied Science Research. 12. 9172-9177. 10.48084/Etasr.5095.
[4] Rahmatulloh, Alam & Nursuwars, Firmansyah. (2017). Implementasi Load Balancing Web Server menggunakan Haproxy dan Sinkronisasi File pada Sistem Informasi Akademik Universitas Siliwangi. Jurnal Teknologi dan Sistem Informasi. 3. 241. 10.25077/TEKNOSI.v3i2.2017.241-248
[5] Safriadi, Safriadi & Rahmadani, Rahmadani. (2024). Analisis Kinerja Load balancing Round Robin Pada Website Skalabel. Journal Of Information System Management (Joism). 5. 227-232. 10.24076/Joism.2024v5i2.1441.
[6] Yang, Z., Zhang, S., Ji, X., & Liu, X. (2018). Research On Cloud Service Quality Control Implementation Based on Improved Load Balance Algorithm. Journal Of Computational Methods in Sciences and Engineering, 18(3), 793–800. Https://Doi.Org/10.3233/Jcm-180830
[7] Nasser, Husain, and Timotius Witono. "Analisis Algoritma Round Robin, Least Connection, Dan Ratio Pada Load Balancing Menggunakan Opnet Modeler." *Informatika: Jurnal Teknologi Komputer dan Informatika*, vol. 12, no. 1, 2016, doi:10.21460/inf.2016.121.455
[8] C. Rawls and M. A. Salehi, "Load Balancer Tuning: Comparative Analysis of HAProxy Load Balancing Methods," High Performance Cloud Computing (HPCC) Lab, University of Louisiana, Lafayette, LA, USA
[9] Saida Helali, "Simulating Network Architectures with Gns3," In Systems and Network Infrastructure Integration: Design, Implementation, Safety and Supervision, Wiley, 2020, Pp.9-25, Doi: 10.1002/9781119779964.Ch2
[10] R. Yusid and N. Callistus Ireneous, "A Novel Algorithm for Efficient Load balancing In Cloud Computing," Asian Journal Research in Computer Science, Vol. 2, Mar. 2024, Doi: 10.9734/Ajrcos/2024/V17i5438
[11] Dodiya, Bindu, and Umesh Kumar Singh. "Malicious Traffic analysis using Wireshark by collection of Indicators of Compromise." Int J Comput Appl 183.53 (2022): 1-6
[12] Jain, G. "Application of snort and wireshark in network traffic analysis." IOP Conference Series: Materials Science and Engineering. Vol. 1119. No. 1. IOP Publishing, 2021
[13] P. Gill, G.J. Garcia, A. Delgado. R.M. Medina, A. Calderon, and P. Marti "Computer Networks Virtualization with GNS3" Computer of Science Research Institute University of Alicante