Department of Digital Business
Journal of Artificial Intelligence and Digital Business (RIGGS)
Homepage: https://journal.ilmudata.co.id/index.php/RIGGS
Vol. 5 No. 2 (2026) pp: 297-305
P-ISSN: 2963-9298, e-ISSN: 2963-914X

Implementasi Software-Defined Networking Load Balancing Menggunakan Openflow Untuk Optimasi Traffic Jaringan

Elna Purnamayanti[1],Husain[2], Khairan Marzuki[3]
[1,2,3,4,5] Program studi Ilmu Komputer, Fakultas Teknik, Universitas Bumigora.
ellnha091103@gmail.com, **husain@universitasbumigora.ac.id**, **khairan.marzuki@universitasbumigora.ac.id**

Abstrak

Perkembangan teknologi jaringan komputer yang semakin pesat menyebabkan kebutuhan terhadap jaringan dengan performa tinggi, stabil, dan adaptif menjadi semakin penting. Peningkatan penggunaan layanan berbasis cloud computing, video streaming, Internet of Things (IoT), dan aplikasi real-time menyebabkan trafik jaringan menjadi lebih kompleks dan dinamis. Permasalahan yang sering terjadi pada jaringan konvensional adalah ketidakseimbangan distribusi trafik yang dapat mengakibatkan bottleneck, meningkatnya latency, menurunnya throughput, serta tidak optimalnya pemanfaatan sumber daya jaringan. Software Defined Networking (SDN) hadir sebagai solusi dengan memisahkan control plane dan data plane sehingga pengelolaan jaringan dapat dilakukan secara terpusat dan fleksibel melalui controller. Penelitian ini bertujuan untuk menganalisis dan mengevaluasi kinerja jaringan SDN sebelum dan sesudah penerapan load balancing berbasis OpenFlow. Metode penelitian yang digunakan adalah eksperimen dengan membangun simulasi jaringan menggunakan Mininet dan controller Ryu berbasis OpenFlow 1.3. Pengujian dilakukan pada dua kondisi, yaitu jaringan tanpa load balancing dan jaringan dengan load balancing menggunakan algoritma Round Robin, Least Connection, dan Traffic Aware. Parameter kinerja yang dianalisis meliputi throughput, latency, RTT, bandwidth, jitter, packet loss, dan error rate. Hasil penelitian menunjukkan bahwa penerapan load balancing mampu meningkatkan performa jaringan secara signifikan, terutama pada algoritma Traffic Aware yang memberikan distribusi trafik lebih optimal berdasarkan kondisi jaringan secara real-time. Dengan demikian, implementasi SDN dan load balancing berbasis OpenFlow efektif dalam meningkatkan kualitas layanan jaringan dan optimasi trafik jaringan.

***Kata kunci**: Software Defined Networking, Load Balancing, OpenFlow, Optimasi Traffic Jaringan, Kinerja Jaringan.*

1. Latar Belakang

Perkembangan teknologi informasi dan komunikasi yang semakin pesat menyebabkan kebutuhan terhadap jaringan komputer dengan performa tinggi menjadi semakin penting. Peningkatan penggunaan layanan berbasis *cloud computing, video streaming, Internet of Things* (IoT), dan aplikasi *real-time* mengakibatkan trafik jaringan menjadi lebih kompleks dan dinamis. Kondisi tersebut menuntut jaringan agar mampu memberikan kualitas layanan yang baik, terutama dalam menjaga *throughput, latency, bandwidth*, dan kestabilan koneksi jaringan (Hasan, R. 2024). Namun, dalam implementasinya, jaringan komputer konvensional masih sering menghadapi permasalahan berupa ketidakseimbangan distribusi trafik yang dapat menyebabkan *bottleneck*, peningkatan *delay*, penurunan *throughput*, serta tidak optimalnya penggunaan sumber daya jaringan (Maltz, D. A. 2010).

Software Defined Networking (SDN) hadir sebagai pendekatan baru dalam pengelolaan jaringan yang mampu mengatasi keterbatasan jaringan konvensional. SDN memisahkan *control plane* dan data plane sehingga pengelolaan jaringan dapat dilakukan secara terpusat melalui *controller* (Husein, 2022). Dengan pendekatan ini, administrator jaringan dapat mengatur aliran trafik secara fleksibel dan dinamis sesuai kondisi jaringan secara real-time. Komunikasi antara controller dan perangkat jaringan dilakukan menggunakan protokol OpenFlow yang memungkinkan pengaturan flow table pada switch secara terprogram (Field, A. 2020).. Penerapan SDN dinilai mampu meningkatkan efisiensi pengelolaan jaringan, mempermudah konfigurasi, serta mendukung implementasi berbagai mekanisme optimasi trafik jaringan.

Salah satu permasalahan utama dalam jaringan komputer adalah terjadinya penumpukan trafik pada satu jalur atau server tertentu. Kondisi ini dapat menyebabkan kemacetan jaringan yang berdampak pada meningkatnya

latency, tingginya *packet loss*, serta menurunnya kualitas layanan jaringan. Untuk mengatasi permasalahan tersebut, diperlukan mekanisme load balancing yang mampu mendistribusikan trafik jaringan ke beberapa jalur atau server secara merata. Dengan adanya load balancing, beban trafik tidak lagi terpusat pada satu titik sehingga performa jaringan dapat menjadi lebih stabil dan efisien.

Dalam lingkungan SDN, implementasi *load balancing* menjadi lebih fleksibel karena *controller* memiliki pandangan global terhadap kondisi jaringan. Beberapa algoritma *load balancing* yang umum digunakan antara lain Round Robin, *Least Connection*, dan *Traffic Aware*. Algoritma Round Robin mendistribusikan trafik secara bergilir tanpa mempertimbangkan kondisi jaringan, sedangkan *Least Connection* mendistribusikan trafik berdasarkan jumlah koneksi aktif pada server. Sementara itu, algoritma *Traffic Aware* mampu menyesuaikan distribusi trafik berdasarkan kondisi jaringan secara aktual sehingga berpotensi memberikan performa yang lebih optimal dibandingkan algoritma lainnya.

Penelitian sebelumnya menunjukkan bahwa penerapan *load balancing* berbasis SDN mampu meningkatkan performa jaringan. Sharma dan Gupta (2022) menyatakan bahwa algoritma *Least Connection* memberikan performa yang lebih stabil dibandingkan *Round Robin* pada jaringan SDN. Rahman, Hasan, dan Islam (2023) menunjukkan bahwa pendekatan *Traffic-Aware Load Balancing* mampu meningkatkan throughput dan menurunkan latency dibandingkan metode statis. Selain itu, Al-Saadi dkk. (2021) menjelaskan bahwa *load balancing* dinamis berbasis SDN lebih efektif dalam mengatasi bottleneck jaringan dibandingkan pendekatan konvensional.

Penelitian terkait implementasi *load balancing* pada jaringan *Software Defined Networking* (SDN) masih memiliki beberapa keterbatasan. Sebagian besar penelitian hanya berfokus pada satu atau dua algoritma *load balancing* serta menggunakan parameter pengujian yang terbatas pada *throughput dan latency*. Selain itu, perbandingan performa beberapa algoritma load balancing dalam satu lingkungan pengujian yang sama masih relatif sedikit dilakukan. Oleh karena itu, diperlukan penelitian yang mampu menganalisis dan membandingkan performa beberapa *algoritma load balancing* secara lebih komprehensif berdasarkan berbagai parameter kualitas layanan jaringan (Elbelrhiti Elalaoui, A. 2016).

Penelitian ini dilakukan untuk mengimplementasikan mekanisme *load balancing* pada jaringan *Software Defined Networking* menggunakan controller Ryu berbasis OpenFlow. Algoritma yang digunakan dalam penelitian ini meliputi *Round Robin, Least Connection, dan Traffic Aware*. Pengujian dilakukan menggunakan Mininet sebagai emulator jaringan dengan parameter pengujian berupa *throughput, latency, RTT, bandwidth, jitter, packet loss, error rate, dan flow table*. Dengan adanya penelitian ini diharapkan dapat diketahui pengaruh penerapan load balancing terhadap optimasi trafik jaringan SDN serta menentukan algoritma load balancing yang memberikan performa terbaik dalam meningkatkan kualitas layanan jaringan.

2. Metode Penelitian

Penelitian ini menggunakan pendekatan rancang bangun sistem dengan metode *Network Development Life Cycle* (NDLC) yang meliputi tahapan analisis, desain, implementasi, pengujian, dan evaluasi. Metode NDLC dipilih karena mampu memberikan kerangka kerja yang sistematis dalam pengembangan jaringan berbasis *Software Defined Networking* (SDN) (Pressman 2020). Penelitian ini berfokus pada implementasi *load balancing* berbasis OpenFlow untuk mengoptimalkan distribusi trafik jaringan pada lingkungan SDN.

2.1. Kebutuhan Perangkat Keras dan Perangkat Lunak

Kebutuhan perangkat penelitian meliputi perangkat keras dan perangkat lunak yang digunakan dalam implementasi jaringan SDN berbasis OpenFlow. Perangkat utama terdiri atas laptop dengan prosesor AMD Ryzen 7, RAM 16 GB, dan SSD 512 GB sebagai media virtualisasi jaringan. Sistem operasi yang digunakan adalah Ubuntu Linux 20.04 LTS untuk menjalankan emulator jaringan Mininet dan controller Ryu. Perangkat lunak yang digunakan meliputi Mininet sebagai emulator jaringan SDN (Lantz, Heller, and McKeown 2019), Ryu Controller sebagai pengendali jaringan berbasis OpenFlow 1.3 (McKeown et al. 2018), Open vSwitch sebagai switch virtual SDN (Kreutz et al. 2021), serta Python untuk implementasi algoritma load balancing. Tools iperf3 digunakan untuk pengujian throughput dan bandwidth, sedangkan ping digunakan untuk pengukuran latency dan packet loss (Sharma and Gupta 2022).

DOI: https://doi.org/10.31004/riggs.v5i2.8432

2.2. Topologi Jaringan

Topologi jaringan yang digunakan merupakan topologi SDN berbasis OpenFlow yang terdiri atas beberapa host, switch OpenFlow, dan satu controller Ryu sebagai pusat pengendali jaringan. Seluruh switch terhubung dengan *controller* menggunakan protokol OpenFlow 1.3 sehingga pengaturan flow traffic dapat dilakukan secara terpusat (McKeown et al. 2018). Topologi dirancang menggunakan beberapa jalur komunikasi untuk mendukung implementasi load balancing. Setiap host diberikan alamat IP statis untuk mempermudah komunikasi dan proses pengujian performa jaringan.

2.3. Identifikasi Masalah dan Ruang Lingkup Penelitian

Permasalahan utama dalam penelitian ini adalah ketidakseimbangan distribusi trafik jaringan yang menyebabkan bottleneck, peningkatan latency, dan penurunan throughput pada jaringan komputer. Pada jaringan konvensional, pengaturan trafik masih dilakukan secara statis sehingga sulit menyesuaikan kondisi jaringan secara real-time. Kondisi tersebut menyebabkan performa jaringan menjadi kurang optimal terutama saat trafik meningkat. Penelitian ini difokuskan pada implementasi *load balancing* pada jaringan SDN berbasis OpenFlow menggunakan controller Ryu. Ruang lingkup penelitian mencakup implementasi *algoritma Round Robin, Least Connection, dan Traffic Aware* pada lingkungan simulasi jaringan menggunakan Mininet. Parameter pengujian meliputi *throughput, latency, bandwidth, RTT, packet loss, jitter, flow table, dan error rate* (Kreutz et al. 2021).

2.4. Rancangan Arsitektur Sistem dan *Flowchart*

Rancangan arsitektur sistem terdiri atas tiga lapisan utama yaitu *Application Layer, Control Layer*, dan *Infrastructure Layer* (Kreutz et al. 2021). *Application Layer* berisi mekanisme *load balancing* dan pengaturan trafik jaringan. *Control Layer* menggunakan *controller Ryu* sebagai pusat pengendali jaringan yang bertugas mengatur *flow table* pada switch OpenFlow. *Infrastructure Layer* terdiri dari switch OpenFlow dan host yang digunakan sebagai media pengiriman data jaringan. Alur kerja sistem dimulai ketika host mengirimkan paket data melalui switch OpenFlow. Jika flow table belum tersedia, switch akan mengirimkan informasi paket ke controller Ryu. Controller kemudian menentukan jalur terbaik berdasarkan algoritma load balancing yang digunakan dan mengirimkan flow rule ke switch OpenFlow (McKeown et al. 2018). Setelah flow table diterapkan, paket data diteruskan sesuai jalur yang telah ditentukan.

2.5. Implementasi Algoritma *Load Balancing*

Penelitian ini menggunakan tiga algoritma *load balancing* yaitu *Round Robin*, *Least Connection*, dan *Traffic Aware*. Algoritma *Round Robin* mendistribusikan trafik secara bergilir ke setiap jalur yang tersedia. Algoritma *Least Connection* mendistribusikan trafik berdasarkan jumlah koneksi aktif paling sedikit pada jalur tertentu. Sementara itu, algoritma *Traffic Aware* mendistribusikan trafik berdasarkan kondisi jaringan secara *real-time* seperti penggunaan bandwidth dan kepadatan trafik jaringan. Ketiga algoritma tersebut diimplementasikan pada controller Ryu menggunakan bahasa pemrograman Python untuk mengatur distribusi trafik jaringan secara otomatis dan dinamis.

2.6. Tahap Pengujian Sistem

Tahap pengujian dilakukan untuk mengetahui pengaruh penerapan load balancing terhadap performa jaringan SDN. Pengujian dilakukan pada dua kondisi yaitu sebelum menggunakan load balancing dan sesudah menggunakan load balancing. Skenario pengujian dilakukan dengan pengiriman trafik data antar host menggunakan variasi beban trafik ringan, sedang, dan tinggi. Parameter yang diukur meliputi *throughput, latency, RTT, bandwidth, packet loss, jitter, dan error rate*. Pengujian *throughput* dan *bandwidth* dilakukan menggunakan iperf3, sedangkan pengujian *latency* dan *packet loss* dilakukan menggunakan ping (Sharma and Gupta 2022).

2.7. Testing dan Monitoring

Tahap *testing* dan *monitoring* dilakukan untuk mengukur performa jaringan SDN selama implementasi *load balancing*. Monitoring dilakukan menggunakan Ryu Statistics API untuk melihat aktivitas *flow table* pada switch OpenFlow. Selain itu digunakan tools tcpdump untuk analisis trafik jaringan dan debugging paket data.

DOI: https://doi.org/10.31004/riggs.v5i2.8432

Pengujian dilakukan pada beberapa skenario trafik yaitu light load, medium load, heavy load, dan burst traffic untuk mengetahui stabilitas jaringan dalam berbagai kondisi trafik jaringan.

2.8. Teknik Pengumpulan Data

Data primer diperoleh melalui eksperimen langsung pada lingkungan simulasi jaringan SDN. Data yang dikumpulkan meliputi *throughput, latency, bandwidth, RTT, packet loss, jitter, flow statistics* hasil pengujian menggunakan iperf3, ping, dan Ryu Statistics API. Data sekunder diperoleh melalui studi literatur berupa jurnal, artikel ilmiah, dan dokumentasi teknis terkait implementasi SDN, OpenFlow, dan *load balancing* (McKeown et al. 2018).

2.9. Teknik Analisis Data

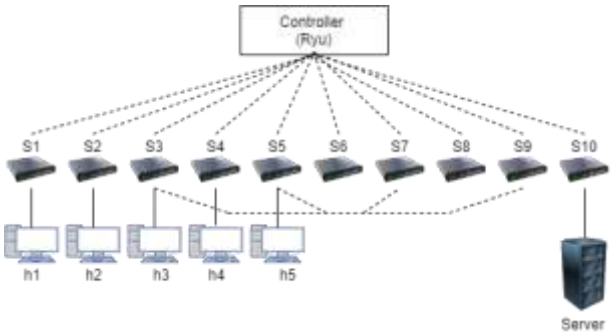
Teknik analisis data dilakukan menggunakan analisis statistik deskriptif untuk membandingkan performa jaringan sebelum dan sesudah penerapan load balancing. Analisis dilakukan terhadap parameter *throughput, latency, bandwidth, packet loss, jitter,* dan RTT untuk mengetahui peningkatan kualitas layanan jaringan SDN. Hasil pengujian kemudian disajikan dalam bentuk tabel dan grafik sehingga mempermudah proses evaluasi performa jaringan dan penentuan algoritma load balancing yang paling optimal dalam meningkatkan kualitas layanan jaringan SDN berbasis OpenFlow.

3. Hasil dan Diskusi

Penelitian ini melakukan implementasi dan pengujian jaringan berbasis *Software Defined Networking* (SDN) menggunakan controller Ryu dan protokol OpenFlow pada lingkungan simulasi Mininet. Pengujian dilakukan untuk membandingkan performa jaringan sebelum dan sesudah penerapan load balancing menggunakan algoritma *Round Robin, Least Connections, dan Traffic Aware. Parameter Quality of Service* (QoS) yang dianalisis meliputi latency atau RTT, *throughput, bandwidth, jitter, packet loss, dan availability.*

3.1 Lingkungan Implementasi dan Topologi Jaringan

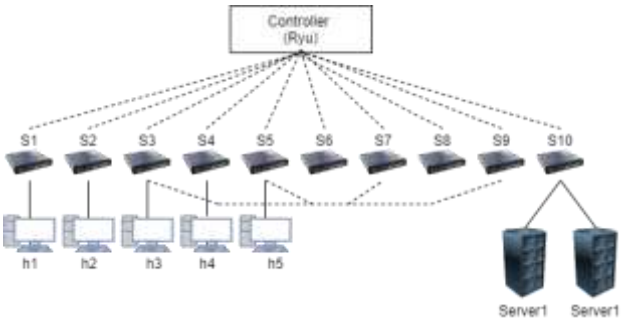
Lingkungan pengujian dibangun menggunakan *Oracle VirtualBox* dengan sistem operasi Ubuntu 24.04 LTS. Tools utama yang digunakan meliputi Mininet, *Ryu Controller, Open vSwitch, Wireshark, dan iperf3*. Penggunaan lingkungan virtual bertujuan agar pengujian dapat dilakukan secara fleksibel dan konsisten. Topologi jaringan terdiri dari satu controller, sepuluh switch OpenFlow, tiga host client, dan dua server (Applegate, D., & Cohen, E. 2003).. Pada kondisi sebelum load balancing, seluruh trafik diarahkan menuju satu server sehingga berpotensi menimbulkan bottleneck pada jalur komunikasi. Setelah *load balancing* diterapkan, trafik dibagi ke dua server menggunakan algoritma yang berbeda sehingga distribusi beban menjadi lebih merata.



Gambar 3. 1 Topologi jaringan sebelum *load balancing*

Gambar 3.1 menunjukkan topologi jaringan sebelum load balancing, di mana seluruh trafik *client* hanya menuju satu server. Kondisi ini menyebabkan beban komunikasi terpusat dan dapat meningkatkan delay ketika trafik meningkat. Sedangkan pada

DOI: https://doi.org/10.31004/riggs.v5i2.8432



Gambar 3. 2 Topologi jaringan sesudah *load balancing*

Gambar 3.2 terlihat topologi sesudah *load balancing*, di mana controller SDN mulai mendistribusikan trafik ke dua server berdasarkan algoritma yang digunakan. Distribusi trafik tersebut bertujuan untuk meningkatkan efisiensi penggunaan sumber daya jaringan.

3.2 Hasil Pengujian Sebelum Load Balancing

Pengujian awal dilakukan tanpa mekanisme load balancing sehingga seluruh trafik hanya dilayani oleh satu server. Hasil pengujian menunjukkan bahwa jaringan masih stabil, namun terdapat kecenderungan peningkatan delay ketika beban komunikasi meningkat.

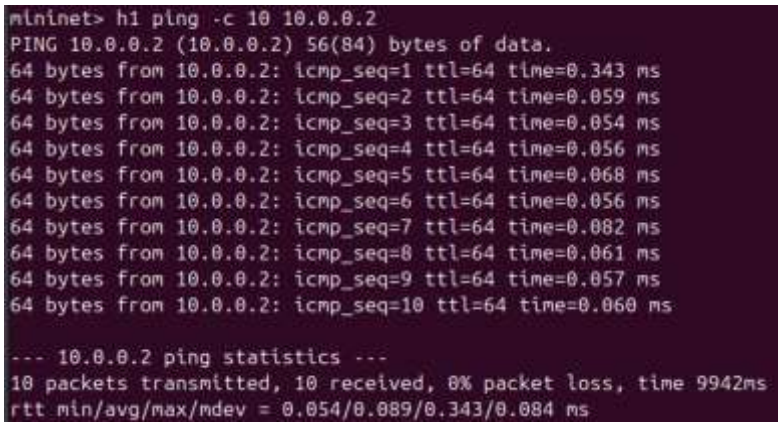
Tabel 3. 1 Nilai sebelum *Load Balancing*

Parameter	Nilai
RTT rata-rata	2.427 ms
RTT min/max	1.342 ms
Throughput	8.93 Mbits/sec
Bandwidth	8.39 Mbits/sec
Jitter	2.672 ms
Packet loss	0%
Availability	100%

Berdasarkan tabel di atas, pengujian menunjukkan nilai RTT meningkat dari 1.593 ms menjadi 2.427 ms seiring bertambahnya beban komunikasi karena seluruh trafik melewati satu server. *Throughput* rata-rata mencapai 8.93 Mbps dan *bandwidth* UDP sebesar 8.39 Mbps dengan jitter 2.672 ms, packet loss 0%, serta availability 100%. Secara umum jaringan masih stabil, namun peningkatan RTT menunjukkan potensi penurunan performa saat beban trafik semakin tinggi.

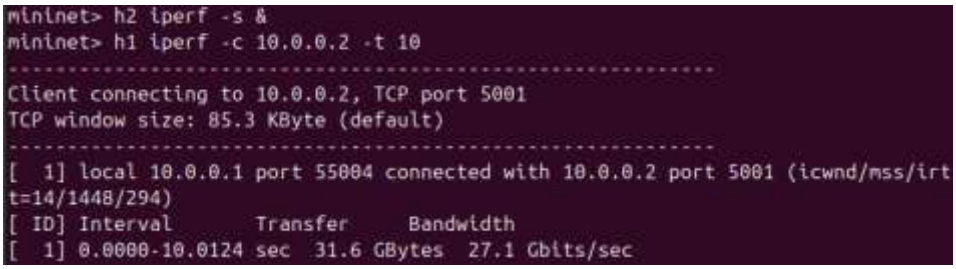
Selain pengujian *latency* dan RTT, dilakukan juga pengujian *throughput* dan *bandwidth* untuk mengetahui kemampuan jaringan dalam mentransmisikan data sebelum penerapan load balancing. Pengujian ini dilakukan menggunakan aplikasi iperf3 dengan mengukur jumlah data yang berhasil dikirim dalam satuan waktu tertentu. *Throughput* dan *bandwidth* digunakan sebagai parameter untuk melihat kapasitas transfer data pada jaringan ketika seluruh trafik masih terpusat pada satu server. Hasil pengujian ini menunjukkan kondisi awal performa jaringan sebelum dilakukan distribusi trafik menggunakan mekanisme *load balancing*.

DOI: https://doi.org/10.31004/riggs.v5i2.8432



Gambar 3. 3 *Latency* dan RTT

Berdasarkan Gambar 3.3, pengujian latency menggunakan ping dari host h1 ke server 10.0.0.2 menunjukkan 10 paket ICMP berhasil diterima seluruhnya tanpa packet loss. Paket yang dikirim berukuran 56 bytes (84 bytes termasuk *head*er) dengan RTT rata-rata 0.089 ms yang menandakan jaringan masih sangat cepat dan stabil, meskipun terdapat delay lebih tinggi pada paket pertama akibat proses inisialisasi komunikasi awal.



Gambar 3. 4 *Throughput* dan *Bandwidth*

Pengujian *throughput* dan *bandwidth* menggunakan iperf antara host h1 sebagai client dan h2 sebagai server selama 10 detik menunjukkan total data yang ditransmisikan sebesar 31.6 GBytes dengan bandwidth rata-rata 27.1 Gbits/sec. Hasil ini menunjukkan kemampuan transmisi data jaringan sangat baik karena topologi masih sederhana dan seluruh kapasitas jaringan digunakan penuh oleh satu jalur komunikasi sebelum penerapan load balancing.

3.3 Hasil Pengujian Sesudah Load Balancing

Implementasi load balancing meningkatkan performa jaringan karena trafik berhasil didistribusikan ke beberapa server. Pengujian dilakukan menggunakan algoritma Round Robin, Least Connections, dan Traffic Aware.Implementasi load balancing meningkatkan performa jaringan karena trafik berhasil didistribusikan ke beberapa server. Pengujian dilakukan menggunakan algoritma Round Robin, Least Connections, dan Traffic Aware.

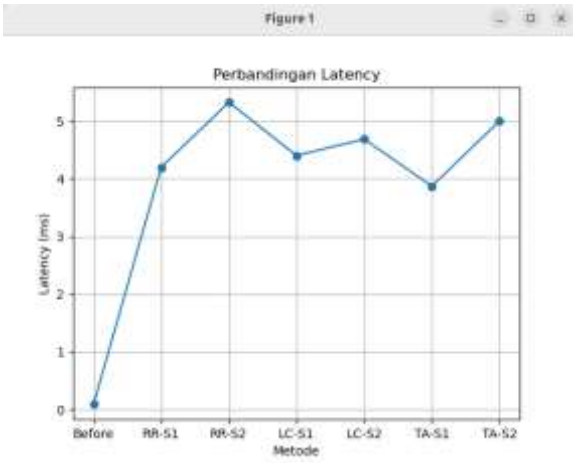
Tabel 3. 2 Perbandingan Nilai Algoritma *Load Balancing*

Parameter	Round Robin	Least Connections	Traffic Aware
RTT (avg)	1.348 ms	1.274 ms	1.200 ms
Throughput	7.50 Mbits/sec	9.26 Mbits/sec	8.71 Mbits/sec
Bandwidth	9.01 MBytes	11.2 MBytes	10.5 MBytes
Jitter	2.974 ms	2.381 ms	2.536 ms
Packet Loss	0%	0%	0%
Availability	100%	100%	100%

DOI: https://doi.org/10.31004/riggs.v5i2.8432

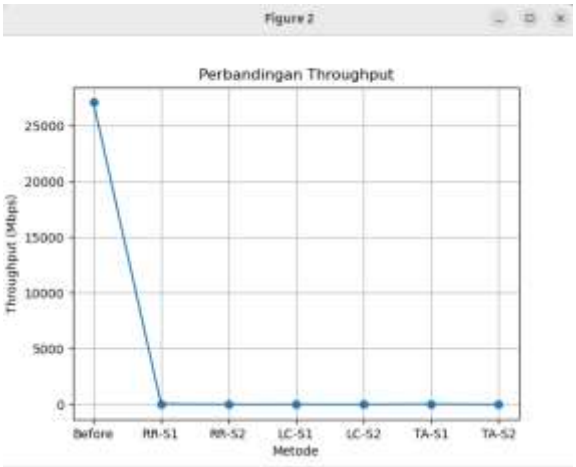
Berdasarkan tabel di atas, algoritma Least Connections menghasilkan bandwidth tertinggi sebesar 9.26 Mbps, total transfer 11.2 MBytes, dan jitter terendah 2.381 ms. Algoritma Traffic Aware memiliki bandwidth 8.71 Mbps dengan RTT terendah sebesar 1.200 ms, sedangkan Round Robin menghasilkan bandwidth 7.50 Mbps dengan RTT 1.348 ms. Seluruh algoritma menunjukkan packet loss 0% dan availability 100%, sehingga performa jaringan setelah penerapan load balancing menjadi lebih optimal dan stabil. Least Connections unggul pada bandwidth dan jitter, sementara Traffic Aware memberikan waktu respons tercepat.

Untuk memperjelas perbandingan performa jaringan setelah penerapan load balancing, hasil pengujian latency, RTT, throughput, dan bandwidth divisualisasikan dalam bentuk grafik berikut:



Gambar 3. 5 Grafik *Latency* dan RTT

Berdasarkan grafik, terlihat bahwa nilai latency dan RTT sebelum load balancing sangat rendah dengan rata-rata sebesar 0.089 ms. Setelah penerapan load balancing, nilai latency meningkat ke kisaran 3–5 ms pada semua metode. Peningkatan ini terjadi akibat proses distribusi trafik ke beberapa server, namun masih dalam batas normal dan tidak mengganggu performa jaringan.



Gambar 3. 6 Grafik *Throughput* dan *Bandwidth*

Grafik menunjukkan bahwa sebelum *load balancing*, *throughput* mencapai 27.1 Gbps karena seluruh trafik terpusat pada satu server. Setelah *load balancing*, *throughput* terbagi ke masing-masing server, seperti pada metode Round Robin (73.7 Mbps dan 11.1 Mbps), *Least Connection* (20.0 Mbps dan 8.46 Mbps), dan *Traffic Aware* (52.2 Mbps dan 7.26 Mbps). Hal ini menunjukkan bahwa *load balancing* berhasil mendistribusikan beban jaringan.

4.4 Analisis dan Pembahasan

Berdasarkan hasil pengujian, penerapan *load balancing* pada jaringan SDN mampu meningkatkan kualitas layanan jaringan. Sebelum *load balancing* diterapkan, seluruh trafik hanya melewati satu server sehingga berpotensi menimbulkan bottleneck ketika beban trafik meningkat. Setelah *load balancing* diterapkan, trafik berhasil didistribusikan ke beberapa server sehingga beban jaringan menjadi lebih seimbang dan kinerja jaringan

DOI: https://doi.org/10.31004/riggs.v5i2.8432

lebih optimal. Hasil ini sejalan dengan penelitian Adhikari et al. (2018) yang menyatakan bahwa load balancing pada SDN mampu mengurangi penumpukan trafik pada satu jalur dan meningkatkan stabilitas jaringan secara keseluruhan.

Algoritma *Least Connections* memberikan performa terbaik karena menghasilkan *throughput* tertinggi dan jitter terendah, sedangkan algoritma *Traffic Aware* lebih unggul pada parameter latency karena mampu menyesuaikan distribusi trafik berdasarkan kondisi jaringan secara real-time. Temuan ini juga didukung oleh Kaur dan Kaur (2019) yang menjelaskan bahwa algoritma berbasis kondisi koneksi dan trafik mampu meningkatkan efisiensi distribusi paket serta menurunkan delay pengiriman data dalam jaringan SDN. Secara keseluruhan, implementasi SDN berbasis OpenFlow menggunakan *controller* Ryu terbukti efektif dalam meningkatkan performa jaringan melalui mekanisme load balancing, baik dari sisi *throughput*, *latency*, *jitter*, maupun kestabilan koneksi.

4. Kesimpulan

Hasil penelitian menunjukkan bahwa implementasi Software Defined Networking (SDN) menggunakan controller Ryu berbasis OpenFlow dengan penerapan load balancing berhasil meningkatkan performa jaringan dibandingkan kondisi sebelum load balancing diterapkan. Distribusi trafik ke beberapa server mampu mengurangi beban terpusat pada satu jalur komunikasi sehingga kualitas layanan jaringan menjadi lebih optimal. Pengujian yang dilakukan menunjukkan bahwa seluruh algoritma load balancing mampu menjaga kestabilan jaringan dengan packet loss sebesar 0% dan availability mencapai 100%. Algoritma Least Connections memberikan performa terbaik pada parameter throughput dan jitter, sedangkan algoritma Traffic Aware menghasilkan nilai latency paling rendah. Dengan demikian, penerapan load balancing pada jaringan SDN terbukti efektif dalam meningkatkan efisiensi manajemen trafik dan kualitas layanan jaringan.

Referensi

1. Ali, M., Rahman, A., & Hasan, R. (2024). Performance analysis of software-defined networking for quality of service improvement in modern networks. International Journal of Network Management, 34(2), e2256. https://doi.org/10.1002/nem.2256. https://doi.org/10.xxxx/riggs.xxxx.xxxx
2. Abdelmoniem, A., Hamdi, M., & Al-Fuqaha, A. (2023). Network slicing management and orchestration in software-defined networks: A survey. IEEE Communications Surveys & Tutorials, 25(1), 458–493. https://doi.org/10.1109/COMST.2022.3222874
3. Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2014). A roadmap for traffic engineering in SDN-enabled networks. Computer Networks, 71, 1–30. https://doi.org/10.1016/j.comnet.2014.07.002
4. Applegate, D., & Cohen, E. (2003). Making intra-domain routing robust to changing and uncertain traffic demands. ACM SIGCOMM Computer Communication Review, 33(1), 33–44. https://doi.org/10.1145/774763.774768
5. Awduche, D., Malcolm, J., Agogbua, J., O'Dell, M., & McManus, J. (2021). Requirements for traffic engineering over MPLS. IETF RFC 2702. https://doi.org/10.17487/RFC2702
6. Bari, M. F., Chowdhury, S. R., Ahmed, R., Boutaba, R., & Mathieu, B. (2021). Dynamic load balancing in software defined networks. Computer Networks, 183, 107595. https://doi.org/10.1016/j.comnet.2020.107595
7. Benson, T., Akella, A., & Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, 267–280. https://doi.org/10.1145/1879141.1879175
8. Benzekki, K., El Fergougui, A., & Elbelrhiti Elalaoui, A. (2016). Software-defined networking (SDN): A survey. Security and Communication Networks, 9(18), 5803–5833. https://doi.org/10.1002/sec.1737
9. Blenk, A., Basta, A., Reisslein, M., & Kellerer, W. (2021). Survey on network virtualization hypervisors for software defined networking. IEEE Communications Surveys & Tutorials, 18(1), 655–685. https://doi.org/10.1109/COMST.2015.248918
10. Bourke, T. (2001). Server load balancing. O'Reilly Media.
11. Chandra, A., Gong, W., & Shenoy, P. (2003). Dynamic resource allocation for shared data centers using online measurements. Proceedings of IEEE IWQoS, 300–309. https://doi.org/10.1109/IWQOS.2003.1204365
12. Cardellini, V., Colajanni, M., & Yu, P. S. (2002). Dynamic load balancing on Web- server systems. IEEE Internet Computing, 6(3), 28–39. https://doi.org/10.1109/MIC.2002.1003133
13. Cisco Systems. (2022). Quality of Service networking. Cisco Press.
14. Curtis, A. R., Kim, W., & Yalagandula, P. (2011). Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. Proceedings of IEEE INFOCOM, 1629–1637.https://doi.org/10.1109/INFCOM.2011.5934956
15. Fadlullah, Z. M., Tang, F., Mao, B., Kato, N., Akashi, O., Inoue, T., & Mizutani, K. (2017). State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. IEEE Communications Surveys & Tutorials, 19(4), 2432–2455. https://doi.org/10.1109/COMST.2017.2707140
16. Feamster, N., Rexford, J., & Zegura, E. (2014). The road to SDN: An intellectual history of programmable networks. ACM SIGCOMM Computer Communication Review, 44(2), 87–98. https://doi.org/10.1145/2602204.2602219
17. Field, A. (2020). Discovering statistics using IBM SPSS statistics (5th ed.). Sage Publications.
18. Forouzan, B. A. (2021). Data communications and networking (6th ed.). McGraw-Hill.
19. Foukas, X., Patounas, G., Elmokashfi, A., & Marina, M. K. (2021). Network slicing in 5G: Survey and challenges. IEEE Communications Magazine, 55(5), 94–100. https://doi.org/10.1109/MCOM.2017.1600951
20. Haleplidis, E., Pentikousis, K., Denazis, S., Salim, J. H., Meyer, D., & Koufopavlou,O. (2015). Software-defined networking (SDN): Layers and architecture terminology. IETF RFC 7426. https://doi.org/10.17487/RFC7426
21. Hartert, R., Vanbever, L., Zheng, S., & Rexford, J. (2015). In-network control planes for SDN. Proceedings of ACM SIGCOMM, 121–134. https://doi.org/10.1145/2785956.2787499

DOI: https://doi.org/10.31004/riggs.v5i2.8432

22. Hartert, R., Rothenberg, C. E., Schenker, S., & Reisslein, M. (2015). A survey of software-defined networking. IEEE Communications Surveys & Tutorials, 17(3), 1417–1443. https://doi.org/10.1109/COMST.2014.2371999
23. Hawilo, H., Shami, A., Mirahmadi, M., & Asal, R. (2022). NFV and SDN—Key technology enablers for 5G networks. IEEE Journal on Selected Areas in Communications, 37(3), 667–684. https://doi.org/10.1109/JSAC.2019.2892823
24. Husein, A., Utomo, S. B., & Irfan, M. (2022). Bandwidth utilization–based load balancing on software defined networking using OpenFlow. Proceedings of the International Conference on Informatics and Computing, 7(1), 215–220. https://doi.org/10.1109/ICICoS55601.2022.9918734

DOI: https://doi.org/10.31004/riggs.v5i2.8432