Ranah Research
Journal of Multidisciplinary Research and Development
E-ISSN: 2655-0865
DINASTI RESEARCH
082170743613 ranahresearch@gmail.com https://jurnal.ranahresearch.com

DOI: https://doi.org/10.38035/rrj.v8i5

Peningkatan Efisiensi Penggunaan *Software Defined Network* Menggunakan Metode *Least Load Path*

Novresa Dwi Yudanti[1], Andi Adriansyah[2]
[1]Universitas Mercu Buana, Jakarta, Indonesia, novresayudanti2911@gmail.com
[2]Universitas Mercu Buana, Jakarta, Indonesia, andi@mercubuana.ac.id

Corresponding Author: novresayudanti2911@gmail.com[1]

Abstract: *Load imbalance is a major problem in Software Defined Network (SDN) architectures, occurring when traffic distribution is uneven so that some links experience congestion while other paths remain underutilized. This study aims to implement the Least Load Path (LLP) method to improve the utilization efficiency of SDN and to compare it with the Shortest Path (SP) and Simple Switch (SS) methods. The study was conducted using the Mininet emulator with a multipath topology of seven OpenFlow switches, the Ryu Controller, and NetworkX. The LLP method monitors OpenFlow port statistics every five seconds, computes the load of each link using the formula Load (bps) = (Δbyte × 8) / time interval as edge weights, and applies Dijkstra's algorithm to find the path with the lowest total weight. Testing was carried out under three load scenarios, measuring packet loss, Round Trip Time (RTT), throughput, and path selection. The results show that LLP maintained 0% packet loss across all scenarios, preserved an average RTT of 27.24 ms under high load (95.84% better than SP which degraded to 655.64 ms), and achieved a throughput of 72.30 Mbps under medium load (36.16% higher than SP and 43.7% higher than SS). LLP was also the only method capable of dynamic path selection. This study confirms that LLP is an effective and adaptive load balancing mechanism for SDN environments.*

Keyword: *Software Defined Network, Least Load Path, load balancing, OpenFlow, Ryu Controller, Mininet, Quality of Service.*

Abstrak: Ketidakseimbangan beban merupakan permasalahan utama pada arsitektur Software Defined Network (SDN), yaitu ketika distribusi trafik tidak merata sehingga sebagian link mengalami congestion sementara jalur lain tidak dimanfaatkan secara optimal. Penelitian ini bertujuan menerapkan metode Least Load Path (LLP) untuk meningkatkan efisiensi penggunaan jaringan SDN dan membandingkannya dengan metode Shortest Path (SP) dan Simple Switch (SS). Penelitian dilakukan menggunakan emulator Mininet dengan topologi multipath tujuh switch OpenFlow, Ryu Controller, dan NetworkX, yang diuji pada tiga skenario beban. Hasil menunjukkan LLP mempertahankan packet loss 0% pada seluruh skenario, RTT rata-rata 27,24 ms pada beban tinggi (95,84% lebih baik dari SP), dan throughput 72,30 Mbps pada beban sedang (43,7% lebih tinggi dari SS). LLP terbukti efektif sebagai mekanisme load balancing yang adaptif pada jaringan SDN.

Kata Kunci: *Least Load Path, load balancing, OpenFlow, Quality of Service, Software Defined Network.*

PENDAHULUAN

Perkembangan layanan digital yang pesat menuntut infrastruktur jaringan yang mampu mengelola trafik dalam volume besar secara efisien, andal, dan adaptif. Arsitektur jaringan tradisional yang memadukan bidang kendali (*control plane*) dan bidang data (*data plane*) di dalam setiap perangkat menyulitkan pengelolaan trafik secara terpusat dan menyeluruh. Keterbatasan ini mendorong hadirnya paradigma *Software Defined Network* (SDN), yang memisahkan *control plane* dari *data plane* sehingga pengendalian jaringan dapat dilakukan secara terprogram dan terpusat melalui *controller*. Pemisahan ini memberikan pandangan global terhadap kondisi jaringan (*global network view*) yang menjadi keunggulan utama SDN dalam pemilihan rute berbasis beban *link*. Kajian penerapan *OpenFlow* pada jaringan *datacenter* menunjukkan bahwa pandangan global melalui *controller* terpusat menjadikan skema pemilihan rute berbasis beban mampu mencapai utilisasi *bandwidth* hingga 97,5% atau 36% lebih tinggi dibanding ECMP (Miguel-alonso, 2023). Survei algoritma *load balancing* pada jaringan heterogen juga menilai arsitektur SDN/NFV ideal sebagai basis *load balancing* karena kemampuannya memantau kondisi jaringan secara menyeluruh (Gures et al., 2022).

Meskipun demikian, efisiensi SDN sangat bergantung pada strategi *routing* dan *load balancing* yang digunakan. Banyak implementasi masih mengandalkan algoritma jalur terpendek konvensional seperti *Dijkstra*, *Bellman-Ford*, maupun OSPF yang hanya mempertimbangkan bobot statis dan mengabaikan kondisi beban *link* secara *real-time*. Kajian komparatif menunjukkan bahwa *Dijkstra* memang efisien dan hemat sumber daya untuk graf *sparse* serta unggul dibanding *Bellman-Ford* pada jaringan tersebut (Alameri et al., 2024). Penerapannya pada *controller* Ryu maupun POX pun dapat berjalan optimal dengan *packet loss* mendekati 0% (Shodiq & Prihanto, 2022). Namun karena algoritma tradisional hanya optimal pada kondisi jaringan sederhana dan kurang mampu menangani persoalan *NP-hard* maupun multi-metrik (Balancing & Optimization, 2024), pemusatan trafik pada jalur terpendek yang sama berpotensi menimbulkan *bottleneck*, kongesti, dan penurunan kualitas layanan ketika beban meningkat. Bahkan modifikasi *cost* pada *Dijkstra* yang hanya berbasis jarak atau utilisasi *bandwidth* saja terbukti menghasilkan latensi lebih tinggi dibanding kombinasi metrik yang turut mempertimbangkan kondisi beban dan latensi (Adiyanto et al., 2024).

Untuk mengatasi keterbatasan tersebut, berbagai pendekatan *load balancing* dinamis telah dikembangkan. Sebagian penelitian menempuh jalur metaheuristik dan kecerdasan buatan, mulai dari *Particle Swarm Optimization* yang dimodifikasi untuk menurunkan latensi dan *packet loss* secara signifikan (Belgaum et al., 2023), *Ant Colony Optimization* hibrida untuk *routing* dinamis pada SDN skala besar (El-hefnawy et al., 2022), algoritma genetik multi-objektif MOCell (Gonzalez-trejo et al., 2022), hingga pendekatan berbasis *deep reinforcement learning* seperti DQQS (Arif et al., 2024). Pendekatan prediktif berbasis *neural network* juga dikembangkan untuk manajemen kongesti (Prasanth & Uma, 2024), sementara penyeimbangan beban di sisi *control plane* ditempuh melalui migrasi *switch* (Cui et al., 2018) dan strategi partisi domain jaringan yang seimbang (Wu et al., 2023). Di sisi lain, terdapat pula pendekatan yang lebih ringan melalui variasi *Round Robin*, seperti *Dynamic Weighted Round Robin* dengan *controller* terdistribusi yang meningkatkan *throughput* sekaligus menurunkan *packet loss* (Shona, 2025).

Beragam pendekatan tersebut secara konsisten membuktikan bahwa mekanisme *routing* yang mempertimbangkan beban jaringan secara dinamis mampu meningkatkan *throughput*, menurunkan *delay*, dan menekan *packet loss* dibanding metode statis (Bano et al., 2021). Hal serupa berlaku pula pada pengembangan jaringan *mesh* dan jalur terpendek berbasis *OpenFlow* (Rizki et al., 2026) serta jaringan IPv6 (Setiawan et al., 2025). Strategi manajemen kapasitas

berbasis *Dijkstra* juga terbukti menekan *packet loss* dan meningkatkan keberhasilan pengiriman paket secara nyata (Rahmah et al., 2025). Kajian evaluasi kualitas jaringan berskala kampus pun menegaskan bahwa parameter *throughput*, *delay*, dan *packet loss* menjadi tolok ukur utama performa jaringan yang layak (Sujaini et al., 2023). Namun demikian, sebagian besar solusi berbasis metaheuristik dan *machine learning* menuntut kompleksitas komputasi yang tinggi, kebutuhan data pelatihan yang besar, atau waktu konvergensi yang panjang, sehingga tidak selalu efisien untuk diterapkan pada jaringan berskala menengah dengan kebutuhan respons cepat.

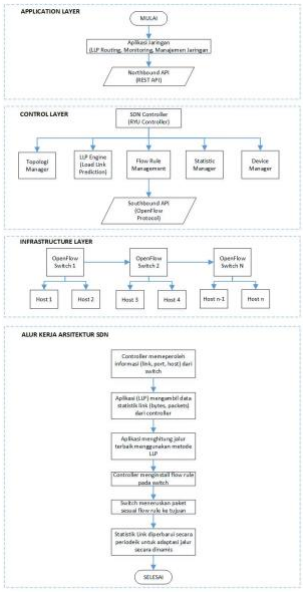
Berpijak pada kesenjangan tersebut, keunikan penelitian ini (*state of the art*) terletak pada penerapan metode *Least Load Path* (LLP) sebagai strategi *routing* yang memilih jalur berdasarkan beban terkecil pada *link* secara adaptif, tanpa memerlukan kompleksitas komputasi setinggi pendekatan metaheuristik maupun *machine learning*. Dengan memanfaatkan pandangan global *controller* SDN, LLP diharapkan mampu mendistribusikan trafik secara lebih merata, menghindari pemusatan beban pada jalur tertentu, serta menjaga kualitas layanan pada kondisi beban yang bervariasi. Pengujian dirancang menggunakan emulator *Mininet* dengan *Ryu Controller* berbasis protokol *OpenFlow* dan pustaka *NetworkX* untuk mengukur peningkatan efisiensi penggunaan *Software Defined Network* melalui parameter *packet loss*, RTT, dan *throughput*. Berdasarkan uraian tersebut, penelitian ini bertujuan menganalisis mekanisme penerapan LLP pada SDN, mengetahui tingkat efisiensi penggunaan jaringan yang dihasilkan, serta membandingkan performa jaringan sebelum dan sesudah penerapan LLP terhadap metode pemilihan jalur konvensional.

METODE

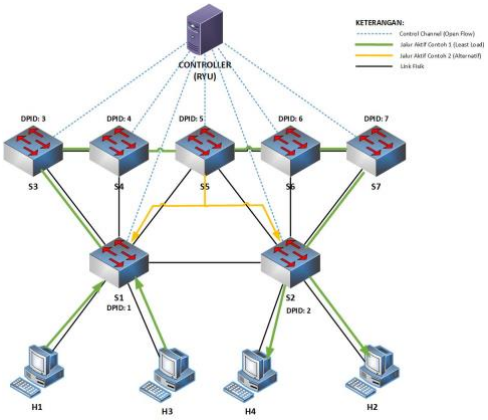
Penelitian ini merupakan penelitian eksperimental kuantitatif dengan pendekatan deskriptif komparatif yang membandingkan performa tiga metode pemilihan jalur pada jaringan *Software Defined Network*, yaitu *Least Load Path* (LLP), *Shortest Path* (SP), dan *Simple Switch* (SS). Objek penelitian adalah jaringan SDN yang dibangun secara virtual menggunakan emulator *Mininet* di atas sistem operasi *Ubuntu*, dengan topologi *multipath* yang terdiri atas satu *Ryu Controller*, tujuh *switch OpenFlow* (s1–s7), dan empat *host* (h1–h4). Setiap *link* dikonfigurasi dengan *bandwidth* 100 Mbps serta *delay* 1 ms pada sisi *host* ke *switch* dan 2 ms pada antar-*switch* untuk merepresentasikan kondisi jaringan nyata. Instrumen perangkat lunak yang digunakan meliputi *Mininet* sebagai emulator, *Ryu Controller* sebagai pengendali, *NetworkX* sebagai pustaka perhitungan graf, serta *iPerf* dan *ping* sebagai alat ukur. Parameter yang diukur secara spesifik adalah *packet loss* (%) melalui pengiriman 100 paket ICMP, *Round Trip Time* (ms) berupa nilai minimum, rata-rata, dan maksimum, *throughput* (Mbps) melalui transfer TCP *iPerf* selama sepuluh detik, serta *path selection* yang diobservasi dari *log controller*.

Prosedur penelitian dimulai dengan menjalankan *Ryu Controller* dan membangun topologi *Mininet*, dilanjutkan registrasi seluruh *host* melalui *ping* awal dan *pingall*. Arsitektur sistem yang dirancang membagi jaringan menjadi tiga lapisan, yaitu *application layer*, *control layer*, dan *infrastructure layer*, sebagaimana ditunjukkan pada Gambar 1. Mekanisme LLP bekerja dengan memantau statistik *port OpenFlow* setiap lima detik, menghitung beban tiap *link* menggunakan rumus Beban (bps) = (Δbyte × 8) / interval waktu, kemudian menjadikan nilai tersebut sebagai bobot *edge* pada graf *NetworkX* dan menerapkan algoritma *Dijkstra* melalui fungsi nx.shortest_path untuk menemukan jalur dengan total bobot terendah. Pengujian dilakukan pada tiga skenario beban, yaitu tanpa beban, beban sedang dengan *background traffic* 50 Mbps, dan beban tinggi dengan empat *stream* paralel 100 Mbps. Pada skenario berbeban, trafik dibangkitkan dengan *iPerf* dan sistem diberi jeda 15 detik agar bobot graf diperbarui sebelum pengukuran RTT dan *throughput* dilakukan. Setiap skenario diulang tiga kali dan nilai yang diambil adalah repetisi paling stabil. Data yang terkumpul dianalisis secara kuantitatif deskriptif komparatif dengan menghitung persentase selisih relatif menggunakan rumus Selisih

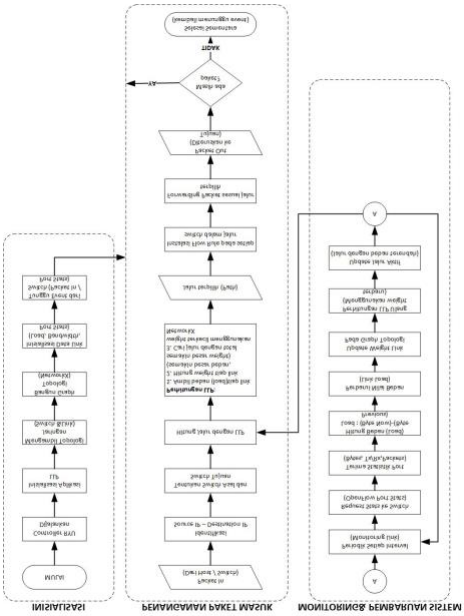
% = |(Nilai Pembanding − Nilai LLP) / Pembanding| × 100%, sedangkan parameter *path selection* dianalisis secara kualitatif berdasarkan konsistensi jalur pada *log controller*.



Gambar 1. Arsitektur Sistem *Software Defined Network*



Gambar 2. Topologi Jaringan *Software Defined Network*



Gambar 3. *Flowchart Controller Least Load Path*

HASIL DAN PEMBAHASAN

Hasil Pengujian

Pengujian dilakukan pada jaringan *Software Defined Network* (SDN) berbasis emulasi menggunakan *Mininet* dengan topologi yang terdiri atas tujuh *switch* dan empat *host*. *Controller* yang diuji meliputi tiga jenis, yaitu *Least Load Path* (LLP), *Shortest Path* (SP), dan *Simple Switch* (SS). Pengujian dilaksanakan pada tiga skenario kondisi beban jaringan, yaitu tanpa beban, beban sedang dengan *background traffic* sebesar 50 Mbps, dan beban tinggi menggunakan empat *stream* paralel dengan *bandwidth* 100 Mbps. Parameter yang diukur meliputi *packet loss*, *Round Trip Time* (RTT), *throughput*, dan *path selection*.

Packet Loss

Packet loss merupakan persentase jumlah paket data yang tidak berhasil diterima oleh *host* tujuan terhadap total paket yang dikirimkan. Hasil pengujian *packet loss* ditunjukkan pada Tabel 1.

Tabel 1. Hasil Pengujian *Packet Loss* (%)

Skenario	LLP	Shortest Path	Simple Switch	Keterangan
Tanpa Beban	**0%**	**0%**	**0%**	**Tidak ada paket hilang**
Beban Sedang	**0%**	**0%**	**0%**	**Tidak ada paket hilang**
Beban Tinggi	**0%**	**0%**	**0%**	**Tidak ada paket hilang**

Sumber: Hasil Pengujian (2026)

Berdasarkan Tabel 1, seluruh *controller* menunjukkan nilai *packet loss* sebesar 0% pada semua skenario pengujian. Hasil ini membuktikan bahwa ketiga *controller* mampu mempertahankan konektivitas jaringan dengan baik tanpa adanya kehilangan paket meskipun dalam kondisi beban tinggi sekalipun. Hal ini mengindikasikan bahwa mekanisme pengiriman paket pada jaringan SDN yang dibangun berjalan dengan stabil pada topologi tujuh *switch* dan empat *host* yang digunakan pada penelitian ini, sehingga perbedaan performa antar-*controller* selanjutnya lebih ditentukan oleh parameter RTT dan *throughput*.

Round Trip Time (RTT)

RTT (*Round Trip Time*) adalah waktu yang dibutuhkan sebuah paket untuk melakukan perjalanan dari *host* pengirim menuju *host* penerima dan kembali ke *host* pengirim. RTT diukur menggunakan perintah *ping* sebanyak 100 paket pada setiap skenario. Hasil pengujian RTT yang meliputi nilai minimum, rata-rata, dan maksimum ditunjukkan pada Tabel 2.

Tabel 2. Hasil Pengujian RTT (ms)

Arah / Metrik	LLP TB	LLP BS	LLP BT	SP TB	SP BS	SP BT	SS TB	SS BS	SS BT
h1→h2 Min	**13,43**	**13,58**	**18,02**	**13,43**	**13,17**	**14,69**	**13,81**	**13,25**	**13,29**
h1→h2 Avg	**21,09**	**31,53**	**27,24**	**20,10**	**15,56**	**655,64**	**15,55**	**19,06**	**15,39**
h1→h2 Max	**145,23**	**112,04**	**89,17**	**85,79**	**22,14**	**2796,76**	**17,99**	**28,29**	**18,61**
h3→h4 Min	**13,20**	**14,43**	**13,93**	**13,50**	**13,75**	**13,41**	**14,00**	**16,37**	**14,50**
h3→h4 Avg	**32,20**	**37,52**	**15,67**	**16,30**	**25,28**	**28,45**	**50,10**	**54,84**	**40,64**
h3→h4 Max	**310,35**	**182,15**	**27,03**	**50,63**	**209,94**	**290,20**	**393,54**	**197,44**	**269,97**

Sumber: Hasil Pengujian (2026). Keterangan: TB = Tanpa Beban, BS = Beban Sedang, BT = Beban Tinggi

Berdasarkan Tabel 2, pada skenario beban tinggi metode *Shortest Path* mengalami lonjakan RTT rata-rata h1→h2 yang sangat signifikan hingga mencapai 655,64 ms dengan RTT

maksimum 2796,76 ms. Kondisi ini terjadi karena *Shortest Path* tidak mampu mendeteksi kepadatan pada *link* s1→s5→s2 sehingga seluruh paket tetap diarahkan melalui jalur yang sama meskipun sudah mengalami kemacetan berat. Sebaliknya, metode *Least Load Path* (LLP) berhasil mempertahankan RTT rata-rata h1→h2 pada beban tinggi sebesar 27,24 ms, jauh lebih rendah dibandingkan *Shortest Path*, yang membuktikan bahwa mekanisme pemilihan jalur berbasis beban pada LLP efektif menghindari *bottleneck* dan menjaga stabilitas latensi jaringan. Adapun *Simple Switch* menunjukkan RTT rata-rata h1→h2 yang relatif rendah dan stabil karena menggunakan jalur terpendek berdasarkan *MAC learning*, namun memiliki RTT rata-rata h3→h4 yang lebih tinggi yaitu 50,10 ms pada kondisi tanpa beban, sehingga performanya tidak konsisten untuk pasangan *host* yang berbeda.

Throughput

Throughput merupakan jumlah data yang berhasil dikirimkan per satuan waktu, diukur dalam satuan Megabit per *second* (Mbps). Pengukuran *throughput* dilakukan menggunakan *iPerf* selama sepuluh detik pada setiap skenario. Hasil pengujian *throughput* ditunjukkan pada Tabel 3.

Tabel 3. Hasil Pengujian *Throughput* (Mbps)

Arah	LLP TB	LLP BS	LLP BT	SP TB	SP BS	SP BT	SS TB	SS BS	SS BT
h1→h2	**62,50**	**61,50**	**53,80**	**43,60**	**58,10**	**49,20**	**63,70**	**56,70**	**60,40**
h3→h4	**61,30**	**72,30**	**65,50**	**57,50**	**53,10**	**43,90**	**65,40**	**50,30**	**72,20**

Sumber: Hasil Pengujian (2026). Keterangan: TB = Tanpa Beban, BS = Beban Sedang, BT = Beban Tinggi

Berdasarkan Tabel 3, metode *Least Load Path* (LLP) menunjukkan *throughput* h3→h4 yang meningkat pada skenario beban sedang, yaitu dari 61,30 Mbps menjadi 72,30 Mbps, karena LLP berhasil mengalihkan *traffic* h3→h4 ke jalur yang lebih ringan sehingga mendapatkan alokasi *bandwidth* yang lebih besar. Sebaliknya, *Simple Switch* mengalami penurunan *throughput* h3→h4 yang cukup signifikan pada beban sedang menjadi hanya 50,30 Mbps, turun 23,10% dari kondisi tanpa beban sebesar 65,40 Mbps, karena tidak dapat mendeteksi kepadatan jaringan sehingga *traffic* h3→h4 tetap menggunakan jalur yang sama dengan *traffic* h1→h2 yang sedang padat. Nilai *throughput* h3→h4 *Simple Switch* pada beban tinggi yang justru naik menjadi 72,20 Mbps terjadi karena *flow rule* lama yang tersimpan di *switch* secara kebetulan mengarahkan trafik melalui jalur berbeda, menandakan performanya bergantung pada kondisi acak dan tidak dapat diandalkan. Sementara itu, *Shortest Path* menunjukkan *throughput* yang tidak konsisten di berbagai skenario karena jalur yang dipilih selalu sama tanpa mempertimbangkan kondisi beban aktual, sehingga ketika suatu *link* padat, *throughput* akan terdegradasi tanpa mekanisme pengalihan.

Path Selection

Path selection merupakan kemampuan *controller* dalam menentukan jalur pengiriman data dari *host* pengirim menuju *host* tujuan. Jalur yang digunakan oleh masing-masing *controller* pada setiap skenario beban dicatat dari *log controller* dan ditunjukkan pada Tabel 4.

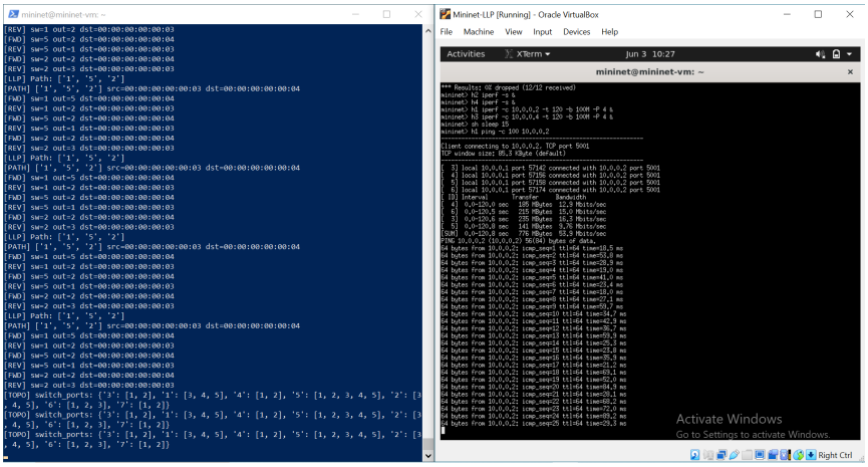
Tabel 4. Hasil *Path Selection*

Skenario	LLP (h1→h2)	Shortest Path (h1→h2)	Simple Switch
Tanpa Beban	**1→5→2**	**1→5→2**	**MAC learning**
Beban Sedang	**1→5→2**	**1→5→2**	**MAC learning**
Beban Tinggi	**1→5→6→2**	**1→5→2**	**MAC learning**

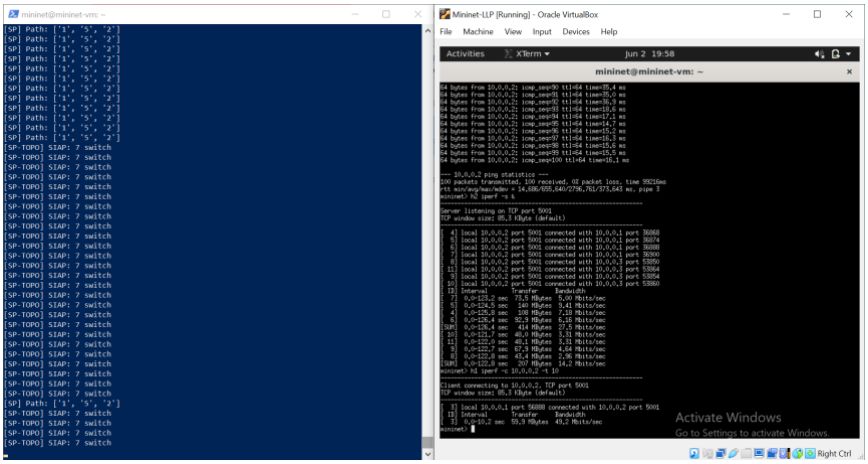
Sumber: Hasil Pengujian (2026)

Berdasarkan Tabel 4, hanya metode *Least Load Path* (LLP) yang mampu mengubah jalur secara dinamis sesuai kondisi beban jaringan. Pada skenario beban tinggi, LLP

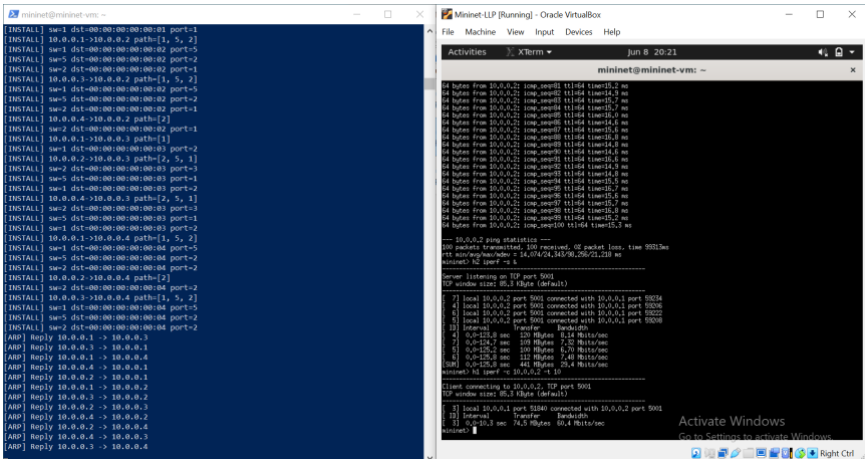
mengalihkan *traffic* h1→h2 dari jalur 1→5→2 menjadi 1→5→6→2 untuk menghindari kemacetan pada *switch* s5. Sebaliknya, *Shortest Path* selalu menggunakan jalur 1→5→2 pada semua skenario karena hanya mempertimbangkan jumlah *hop* tanpa memperhatikan beban *link* aktual, sedangkan *Simple Switch* tidak memiliki mekanisme *path calculation* sehingga jalur tidak dapat ditentukan secara eksplisit karena bergantung pada proses *MAC learning*. Perbedaan inilah yang menjadikan LLP sebagai satu-satunya metode dengan kemampuan adaptasi jalur yang dinamis. Bukti pengukuran RTT untuk masing-masing metode pada kondisi beban tinggi ditunjukkan pada Gambar 4, Gambar 5, dan Gambar 6.



Gambar 4. Pengukuran RTT LLP h1→h2 pada Beban Tinggi



Gambar 5. Pengukuran RTT *Shortest Path* h1→h2 pada Beban Tinggi



Gambar 6. Pengukuran RTT *Simple Switch* h1→h2 pada Beban Tinggi

Least Load Path vs Shortest Path

Tabel 5. Perbandingan LLP vs *Shortest Path*

Aspek	LLP	Shortest Path	Selisih	Keterangan
RTT avg BT h1→h2 (ms)	**27,24**	**655,64**	**628,40**	LLP 95,84% lebih baik
RTT avg BS h3→h4 (ms)	**37,52**	**25,28**	**-12,24**	SP 48,4% lebih baik
Throughput BS h3→h4 (Mbps)	**72,30**	**53,10**	**+19,20**	LLP 36,2% lebih baik
Throughput BT h1→h2 (Mbps)	**53,80**	**49,20**	**+4,60**	LLP 9,4% lebih baik
Adaptasi Path	**Dinamis**	**Statis**	**-**	LLP unggul signifikan
Packet Loss	**0%**	**0%**	**0%**	Tidak ada perbedaan

Sumber: Hasil Pengujian (2026)

Berdasarkan Tabel 5, pada indikator RTT, *Least Load Path* (LLP) menunjukkan keunggulan yang signifikan pada skenario beban tinggi dengan RTT rata-rata h1→h2 sebesar 27,24 ms dibandingkan *Shortest Path* yang mencapai 655,64 ms, atau peningkatan sebesar 95,84%. Lonjakan RTT pada *Shortest Path* terjadi karena seluruh *traffic* dipaksakan melalui satu jalur yang sudah padat, sementara LLP secara otomatis mengalihkan sebagian *traffic* ke jalur alternatif melalui *switch* s6. Pada indikator *throughput*, LLP menunjukkan konsistensi yang lebih baik di seluruh skenario, dengan *throughput* h3→h4 pada beban sedang mencapai 72,30 Mbps, lebih tinggi 36,16% dibandingkan *Shortest Path* sebesar 53,10 Mbps.

Perbedaan paling mendasar antara LLP dan *Shortest Path* terletak pada kemampuan adaptasi jalur. LLP mengubah jalur h1→h2 pada beban tinggi dari 1→5→2 menjadi 1→5→6→2 sebagai respons terhadap peningkatan bobot *link*, sedangkan *Shortest Path* tetap menggunakan jalur 1→5→2 pada semua kondisi beban. Hal inilah yang menyebabkan degradasi performa yang sangat besar pada *Shortest Path* saat beban tinggi. Dengan demikian, LLP terbukti lebih unggul karena memanfaatkan jalur berkapasitas lebih besar berdasarkan kondisi beban aktual yang dipantau setiap lima detik, sesuai dengan temuan bahwa *routing* dinamis lebih efektif dibanding metode statis (Bano et al., 2021).

Least Load Path vs Simple Switch

Tabel 6. Perbandingan LLP vs *Simple Switch*

Aspek	LLP	Simple Switch	Selisih	Keterangan
RTT avg BT h1→h2 (ms)	**21,09**	**15,55**	**-5,54**	SS 35,6% lebih baik
RTT avg BS h3→h4 (ms)	**32,20**	**50,10**	**+17,90**	LLP 35,7% lebih baik
Throughput BS h3→h4 (Mbps)	**72,30**	**50,30**	**+22,00**	LLP 43,7% lebih baik
Throughput BT h1→h2 (Mbps)	**65,50**	**72,20**	**-6,70**	SS 9,28% lebih tinggi
Adaptasi Path	**Dinamis**	**-**	**-**	LLP unggul signifikan
Packet Loss	**0%**	**0%**	**0%**	Tidak ada perbedaan

Sumber: Hasil Pengujian (2026)

Berdasarkan Tabel 6, pada indikator RTT, *Simple Switch* menunjukkan RTT rata-rata h1→h2 yang lebih rendah pada kondisi tanpa beban yaitu 15,55 ms dibandingkan LLP sebesar 21,09 ms. Namun, *Simple Switch* mengalami RTT rata-rata h3→h4 yang lebih tinggi yaitu 50,10 ms dibandingkan LLP sebesar 32,20 ms pada kondisi yang sama. Perbedaan ini menunjukkan bahwa *Simple Switch* tidak selalu menghasilkan performa optimal untuk semua pasangan *host* karena jalur ditentukan hanya berdasarkan *MAC learning* tanpa kalkulasi beban.

Pada indikator *throughput* h3→h4 beban sedang, LLP menghasilkan 72,30 Mbps sedangkan *Simple Switch* hanya 50,30 Mbps, sehingga LLP unggul 43,7%. Meskipun *Simple*

Switch menunjukkan *throughput* h1→h2 lebih tinggi (72,20 Mbps) pada beban tinggi, capaian tersebut bukan hasil optimasi jalur melainkan karena *flow rule* yang tersimpan secara kebetulan mengarahkan *traffic* h1→h2 dan h3→h4 melalui jalur berbeda sehingga tidak terjadi kompetisi *bandwidth*. Hal ini menegaskan bahwa performa *Simple Switch* pada beban tinggi bersifat tidak dapat diprediksi. Sebaliknya, LLP secara aktif memantau statistik *port* setiap lima detik dan membebani bobot jalur berdasarkan beban aktual, sehingga mampu merespons perubahan kondisi jaringan secara dinamis dan mendistribusikan beban lebih merata ke seluruh jalur yang tersedia.

Efektivitas *Load Balancing*

Efektivitas *load balancing Least Load Path* (LLP) dapat dinilai dari tiga aspek utama, yaitu kemampuan mendeteksi beban, kemampuan adaptasi jalur, dan dampak terhadap performa jaringan secara keseluruhan. Kemampuan mendeteksi beban diimplementasikan melalui mekanisme monitoring statistik *port OpenFlow* setiap lima detik. Nilai beban dihitung berdasarkan selisih *byte* yang dikirimkan antar-interval monitoring, kemudian dikonversi ke satuan bps dan digunakan sebagai bobot *edge* pada graf topologi *NetworkX*. Mekanisme ini terbukti efektif dalam mendeteksi peningkatan beban pada *link* s1→s5→s2 dan memicu pemilihan jalur alternatif pada skenario beban tinggi.

Kemampuan adaptasi jalur LLP terbukti dari perubahan jalur h1→h2 pada skenario beban tinggi dari 1→5→2 menjadi 1→5→6→2. Perubahan ini merupakan respons langsung terhadap peningkatan nilai *weight* pada *edge* yang menghubungkan s1→s5 akibat tingginya beban *traffic*. Kemampuan adaptasi ini merupakan keunggulan utama LLP dibandingkan kedua *controller* pembanding yang tidak memiliki mekanisme serupa.

Temuan penelitian (*novelty*) ini menegaskan bahwa LLP mampu berperan sebagai mekanisme *load balancing* yang ringan sekaligus adaptif pada jaringan *Software Defined Network*. Dampaknya terlihat jelas pada skenario beban tinggi, di mana LLP berhasil mencegah degradasi performa yang ekstrem sehingga RTT rata-rata tetap berada di kisaran 15–32 ms pada semua skenario, sementara *Shortest Path* melonjak hingga 655,64 ms dan *Simple Switch* menunjukkan ketidakkonsistenan pada berbagai pasangan *host*. Dengan tetap mempertahankan *packet loss* 0%, hal ini membuktikan bahwa LLP efektif sebagai solusi *load balancing* adaptif tanpa menuntut kompleksitas komputasi tinggi sebagaimana pendekatan metaheuristik maupun *machine learning*.

KESIMPULAN DAN SARAN

Berdasarkan hasil penelitian mengenai peningkatan efisiensi penggunaan *Software Defined Network* menggunakan metode *Least Load Path* (LLP) pada topologi *multipath* tujuh *switch* dan empat *host* berbasis emulasi *Mininet* dengan *Ryu Controller*, dapat disimpulkan bahwa mekanisme LLP berhasil diimplementasikan melalui tahapan inisialisasi *controller*, penanganan paket masuk, serta pemantauan dan pembaruan beban jaringan secara periodik setiap lima detik menggunakan rumus Beban (bps) = (Δbyte × 8) / interval waktu sebagai bobot *edge* dan algoritma *Dijkstra* untuk menemukan jalur berbobot terendah. Penerapan LLP terbukti meningkatkan efisiensi penggunaan jaringan secara signifikan melalui distribusi trafik yang merata dan adaptasi jalur yang dinamis, ditunjukkan dengan pengalihan *traffic* h1→h2 dari jalur s1→s5→s2 ke s1→s5→s6→s2 pada beban tinggi, RTT rata-rata yang stabil di angka 27,24 ms, peningkatan *throughput* h3→h4 pada beban sedang dari 61,30 Mbps menjadi 72,30 Mbps, serta *packet loss* 0% pada seluruh skenario. Dibandingkan metode konvensional, LLP unggul 95,84% pada RTT beban tinggi terhadap *Shortest Path*, 36,16% pada *throughput* beban sedang, dan 43,7% terhadap *Simple Switch*, sekaligus menjadi satu-satunya metode dengan kemampuan *path selection* yang dinamis dan adaptif, sehingga terbukti efektif sebagai mekanisme *load balancing* pada jaringan *Software Defined Network*.

REFERENSI

Alameri, I., Komarkova, J., Al-hadhrami, T., Yahya, A. E., & Gharbi, A. (2024). Optimizing Connections: Applied Shortest Path Algorithms for MANETs. https://doi.org/10.32604/cmes.2024.052107

Arif, F., Khan, N. A., Iqbal, J., Karim, F. K., Innab, N., & Mostafa, S. M. (2024). DQQS: Deep Reinforcement Learning-Based Technique for Enhancing Security and Performance in SDN-IoT Environments. *12*(May).

Balancing, L., & Optimization, D. (2024). Optimization Algorithms in SDN: Routing, Load Balancing, and Delay Optimization. *Applied Sciences*.

Bano, M., Qayyum, A., Naveed, R., Rais, B., & Gilani, S. S. A. (2021). Soft-Mesh: A Robust Routing Architecture for Hybrid SDN and Wireless Mesh Networks. *IEEE Access, 9*, 87715–87730. https://doi.org/10.1109/ACCESS.2021.3089020

Belgaum, M. R., Musa, S., Ali, F., Alam, M. M., Alansari, Z., Soomro, S., Mohd, M., & Ud, S. U. (2023). Self-Socio Adaptive Reliable Particle Swarm Optimization Load Balancing in Software-Defined Networking. *11*(August).

Cui, J., Lu, Q., Zhong, H., Tian, M., & Liu, L. (2018). A Load-balancing Mechanism for Distributed SDN Control Plane Using Response Time. *IEEE Transactions on Network and Service Management, PP*(October), 1. https://doi.org/10.1109/TNSM.2018.2876369

El-hefnawy, N. A., Raouf, O. A., & Askr, H. (2022). Dynamic Routing Optimization Algorithm for Software Defined Networking. https://doi.org/10.32604/cmc.2022.017787

Gonzalez-trejo, J. E., Rivera-rodriguez, R., Tchernykh, A., Lozano-rizk, J. E., Villarreal-reyes, S., Galaviz-mosqueda, A., & Compean, J. L. G. (2022). A Novel Strategy for Computing Routing Paths for Software-Defined Networks Based on MOCell Optimization. *Applied Sciences*.

Gures, E., Shayea, I., Ergen, M., El-saleh, A. A., & Member, S. (2022). Machine Learning-Based Load Balancing Algorithms in Future Heterogeneous Networks: A Survey. *IEEE Access, 10*, 37689–37717. https://doi.org/10.1109/ACCESS.2022.3161511

Miguel-alonso, J. (2023). A Research Review of OpenFlow for Datacenter Networking. *11*(December 2022).

Prasanth, L. L., & Uma, E. (2024). A computationally intelligent framework for traffic engineering and congestion management in software-defined network. *EURASIP Journal on Wireless Communications and Networking*. https://doi.org/10.1186/s13638-024-02392-2

Rahmah, N. S., Hamami, F., & Yanu, R. (2025). Capacity Management Strategy for Handling Traffic Load with Dijkstra Algorithm at PT XYZ. *18*(1).

Rizki, A. T., Coastera, F. F., & Program, I. S. (2026). Comparative Study of Johnson and Bellman-Ford for Shortest Path in OpenFlow SDN. *10*(3), 1243–1249.

Setiawan, S., A, I. K., & Pranata, M. (2025). Analisis Performansi OSPFv3 Pada Jaringan IPv6 Menggunakan Free Range Routing. *15*(2), 99–109. https://doi.org/10.22441/incomtech.v15i2.21442

Shodiq, F. F., & Prihanto, A. (2022). Analisis Perbandingan Performansi Kontroler RYU Dan POX Berbasis Software Defined Network (SDN) Pada Routing OSPF. *03*, 216–223.

Shona, M. (2025). Design and Deployment of a Dynamic Weighted Round-Robin SDN Load Balancing Mechanism with Distributed Controllers. *15*(6), 30260–30266.

Sujaini, H., Imansyah, F., Sholva, Y., Hadary, F., Dolorosa, E., & Ihwan, A. (2023). Evaluasi Kinerja Internet Kampus Universitas Tanjungpura dengan Analisis Quality of Service dan User Acceptance Test. *9*(1), 89–95.

Wu, Y., Du, J., & Ni, D. (2023). Balanced Domain Partitioning for Software Defined Networks. *IEEE Access, 11*(December 2022), 6467–6476. https://doi.org/10.1109/ACCESS.2023.3237733