PENGEMBANGAN MEKANISME *LOAD BALANCING* MENGGUNAKAN MODIFIKASI *DIJKSTRA* PADA JARINGAN SDN *DATA PLANE FAT TREE*

Muhammad Fattahilah Rangkuty[1)], Royyana Muslim Ijtihadie[2)], dan Tohari Ahmad[3)]
[1, 2, 3)] Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia 60111
e-mail: fattahilah.rangkuty@gmail.com[1)], roy@if.its.ac.id[2)], tohari@if.its.ac.id[3)]

ABSTRAK

SDN adalah pendekatan jaringan komputer yang memungkinkan administrator jaringan untuk mengelola layanan jaringan melalui abstraksi fungsionalitas pada tingkat yang lebih tinggi, dengan memisahkan sistem yang membuat keputusan tentang di mana traffic dikirim (control plane), kemudian meneruskan traffic ke tujuan yang dipilih (data plane). SDN dapat memiliki permasalahan pada kemacetan jaringan, latensi yang tinggi, dan penurunan throughput dikarenakan ketidakseimbangan alokasi traffic pada link yang tersedia, sehingga dibutuhkan satu metode load balancing. Teknik ini membagi seluruh beban secara merata pada setiap komponen jaringan pada jalur atau path yang menghubungkan data plane dan S-D (Source Destination) host. Konsep Least Loaded Path (LLP) yang kami ajukan merupakan pengembangan Dijkstra, melakukan pemilihan best path dengan mencari jalur terpendek dan beban traffic terkecil, beban traffic terkecil (minimum cost) didapatkan dari penjumlahan tx dan rx data pada switchport data plane yang terlibat dalam pengujian, hasil ini yang kemudian akan ditetapkan sebagai best path dalam proses load balancing.

Kata kunci: *Data plane, kontroler SDN, load balancing, openflow, software defined networking.*

DEVELOPMENT OF LOAD BALANCING MECHANISMS IN SDN DATA PLANE FAT TREE USING MODIFIED DIJKSTRA'S ALGORITHM

Muhammad Fattahilah Rangkuty[1)], Royyana Muslim Ijtihadie[2)], and Tohari Ahmad[3)]
[1, 2, 3)] Department of Informatics, Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia 60111
e-mail: fattahilah.rangkuty@gmail.com[1)], roy@if.its.ac.id[2)], tohari@if.its.ac.id[3)]

ABSTRACT

SDN is a computer network approach that allows network administrators to manage network services through the abstraction of functionality at a higher level, by separating systems that make decisions about where traffic is sent (control plane), then forwarding traffic to the chosen destination (data plane). SDN can have problems with network congestion, high latency, and decreased throughput due to unbalanced traffic allocation on available links, so a load-balancing load method is needed. This technique divides the entire load evenly on each component of the network on the path or path that connects the data plane and S-D (Source Destination) host. The Least Loaded Path (LLP) of our proposed concept, which is a Dijkstra development, selects the best path by finding the shortest path and the smallest traffic load, the smallest traffic load (minimum cost) obtained from the sum of tx and rx data in the switchport data plane involved in the test, this result which will then be determined as the best path in the load balancing process.

***Keywords**: Data plane, load balancing, openflow, SDN controller, software defined networking.*

I. PENDAHULUAN

KONTROLER merupakan salah satu komponen SDN yang menjadi pusat pengendali, penentu kebijakan jaringan, dan pengaturan *flow table* pada *switch* atau *data plane*, dengan menggunakan protokol *OpenFlow*. Kontroler juga berisikan aplikasi atau perangkat lunak yang mengendalikan trafik pada *switch* atau *data plane*. Melalui kontroler, *administrator* jaringan menguraikan aturan yang mendefinisikan perutean paket pada perangkat jaringan. Semua komunikasi antara perangkat jaringan dan pengontrol dilakukan melalui protokol yang memiliki fitur keamanan yang disebut *OpenFlow*.

Menurut [1], algoritma *Dijkstra* memeperhitungkan bobot *link* saat mencari jalur terpendek, sementara penelitian yang dilakukan oleh [2], penggunaan *Djikstra* dimodifikasi, tidak hanya menggunakan bobot pada *link* namun

ditambah penggunaan bobot *node*, untuk melakukan estimasi *cost* dari jalur dari *source node* ke *destination node*, dengan demikian dapat mengurangi *end-to-end delay* dan mendapatkan kualitas transmisi yang lebih baik.

Menurut [3], salah satu model topologi yang digunakan pada jaringan SDN adalah *fat tree*, *fat tree* merupakan topologi jaringan tipikal *data center* yang banyak digunakan untuk *high availability* dan *scalability*. *fat tree* dapat mendukung jaringan *High Performance Computing* dengan model *cluster* dan pada jaringan *data center*, salah satu kelebihan *fat tree* adalah skalabilitas dan jalur yang bervariasi atau beragam, hal ini memberikan *alternative path* dalam *load balancing* antara *source* dan *destination*, untuk menghindari *congestion* dan peningkatan penggunaan *bandwidth*. Sementara [4] menyebutkan *Fat tree* telah digunakan pada jaringan *data center* berbasis *OpenFlow* SDN untuk menerapkan *load balancing* dengan dukungan *multipath*, yang akan meningkatkan performa dan menurunkan *latency*.

Menurut [5], untuk mencapai suatu *load balancing,* diperlukan pendistribusian *load* secara merata pada beberapa *path*. *Best path* disini dapat diartikan sebagai *path* yang memiliki *load* paling kecil dari semua *path* yang ada, atau bisa juga disebut *path* terpendek antara *source* dan *destination host*. *Path* dengan *load* paling kecil atau *least loaded path* dapat dipilih menjadi jalur yang akan dilewati data sehingga akan meningkatkan *transfer rate data* dan mengurangi *latency*.

Pada tahun 2018 [6] menjelaskan, untuk mencapai persyaratan *Quality of Service* (QoS), *Load Balancing* (LB) pada SDN harus dipertimbangkan untuk mengatasi sumber daya jaringan yang terbatas. Beberapa mekanisme LB telah ditinjau, mereka memberikan manfaat dan kelemahan mekanisme LB pada SDN secara sistematis berdasarkan pada dua faksi, deterministik dan non-deterministik.

Pada tahun 2016, [7] melakukan penelitian *load balancing* pada *data center* dengan metode *round robin* yang diterapkan pada kontroler SDN, untuk mengurangi *cost* dan efisiensi dari penggunaan *load balancer vendor* tertentu. Namun algoritma *round robin* memiliki kekurangan dalam pemilihan *path* atau *link* yang akan digunakan untuk mengirim data, sehingga dapat menimbulkan kemacetan jaringan atau *link overloading*. Penelitian *load balancing* dengan menggunakan algoritma *Djikstra* telah dilakukan, dengan menemukan *shortest path* dari satu titik ke titik yang lain, pada penelitian ini hanya mempertimbangkan jalur terpendek dalam melakukan transfer data, tanpa melihat banyaknya trafik yang sedang melewati jalur terpendek tersebut [8].

Dalam melakukan transmisi paket data antar host pada jaringan SDN, sering terjadi *link overloading* yang disebabkan pemilihan jalur yang tidak sesuai. Pemilihan jalur dibutuhkan untuk mencapai *load balancing* pada jaringan *data plane*, oleh karena itu dibutuhkan suatu mekanisme dalam pemilihan jalur dalam pengiriman paket data untuk menghindari *link overloading.* Penelitian ini menggunakan *load balancing* dengan *flow* dinamis untuk menentukan rute *flow* lain yang mungkin, dan paket data dibagi secara merata antara jalur lain yang memungkinkan untuk dilewati atau jalur yang ditentukan yang memiliki *load* paling sedikit.

Penelitian ini melakukan pengembangan mekanisme *load balancing* dengan *path selection* untuk mencapai *load balancing* dengan algoritma *least loaded path* untuk mendapatkan *load balancing* pada jaringan SDN yang disimulasikan pada jaringan *fat tree topology*. Metode yang dirancang akan meningkatkan performa jaringan SDN, dalam hal peningkatan *transfer rate data*, penggunaan *bandwidth* yang efisien, dan mengurangi *latency*. *Least loaded path* akan bertindak reaktif pada *data plane*, dengan tidak menghiraukan *flow* ketika terdeteksi kelebihan beban pada peralatan jaringan. Proposal ini mengusulkan metode *selection path* untuk *load balancing* dengan *least loaded path* untuk memberikan alternatif dalam pemilihan mekanisme *load balancing*, terutama pada jaringan SDN dengan *fat tree topology*. Untuk pengaturan eksperimental, *controller Floodlight*, *emulator* jaringan *Mininet*, dan analisis paket *Wireshark* digunakan.

Tujuan dari penelitian ini adalah untuk menerapkan algoritma *Least Loaded Path* (LLP) sebagai modifikasi dari *Dijkstra* untuk mencapai *load balancing* dalam jaringan SDN *data plane* menggunakan *topologi fat tree*. Kami fokus pada *load balancing* jaringan *data plane*, yang memiliki masalah dalam mendistribusikan arus di jalur tunggal yang padat dan tidak memiliki rute alternatif untuk *flow*.

Makalah ini disusun sebagai berikut: studi literatur di Bagian 2. Bagian 3 mendefinisikan usulan metode. Bagian 4 menjelaskan skenario uji coba. Bagian 5 menjelaskan hasil dan pembahasan, dan bagian 6 menjelaskan kesimpulan dan penelitian kedepan.

II. STUDI LITERATUR

A. Protokol OpenFlow

Pada standar SDN, protokol *OpenFlow* didefinisikan sebagai salah satu protokol komunikasi yang memungkinkan kontroler SDN untuk berinteraksi dengan infrastruktur jaringan atau *switch* dari berbagai macam *vendor*. Dengan menggunakan protokol *OpenFlow*, jaringan dapat diadaptasikan menjadi lebih baik dengan melakukan pengaturan jaringan sesuai kebutuhan penggunanya. Kontroler SDN bekerja sebagai inisiator untuk melakukan

instalasi *flow* pada *switch* atau *router* untuk mengimplementasikan fungsi jaringan, seperti *forwarding* data, kontrol *flow*, dll.

OpenFlow adalah salah satu protokol dari standar kontroler SDN yang paling popular. *OpenFlow* dikembangkan oleh insinyur-insinyur dari universitas *Stanford*, yang memungkinkan bagi mereka untuk melakukan eksperimen pada jaringan kampus tanpa mengganggu konfigurasi jaringan yang sedang berjalan. *OpenFlow* memungkinkan *switch* membuat VLAN (*Virtual Local Area Networks*) pada konfigurasi jaringan yang sudah ada, dan memisahkan trafik yang bersifat uji coba [9].

Flow table merupakan database yang berisikan *flow entries* yang berkaitan dengan *actions* untuk memerintahkan switch untuk menetapkan beberapa action pada flow tertentu [10].

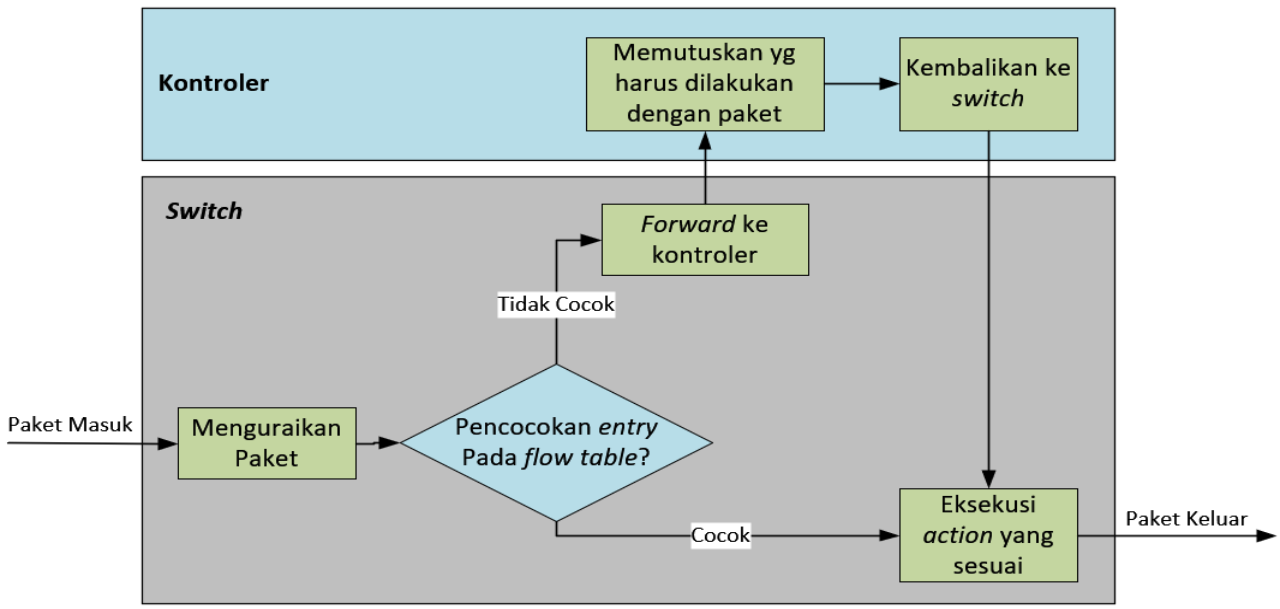
Rule-rule ini akan menciptakan kondisi *match* dan *Action*. Sebuah *rule* yang memiliki semua nilai pada *field* nya disebut *exact rule* dan ketika tidak ada kondisi pada satu atau lebih *field* nya disebut *wildcard rule*. Kondisi *match* memutuskan *rule* mana yang akan diaplikasikan pada paket dan *Action* akan mendefinisikan bagaimana paket akan di tangani. *Match field* dari paket *OpenFlow* dan alur hubungan antara kontroler dan *OpenFlow switch* ditunjukkan pada Gambar 1 dan Gambar2.

Pada Gambar 1 menunjukkan, *match field* dari suatu *rule* pada paket *OpenFlow*. *Rule* merupakan sebuah *field* yang digunakan untuk mencocokan *header* paket. Paket yang masuk pertama dicocokan dengan nomor *in-port* nya, VLAN ID hingga tujuan *port* TCP atau UDP. User dapat mendefinisikan *flow entry* melalui kontroler dimana kemudian dimasukkan pada TCAM (*Ternary Content Addressable Memory*). Paket ini meng-enkapsulasi *Ehernet*, IP, TCP, dan protokol lainnya. Setiap kontroler, sebagai contoh *Ryu* dan *Floodlight* menangani enkapsulasi ini dengan cara yang berbeda tergantung bahasa pemrograman yang digunakan pada masing-masing kontroler.

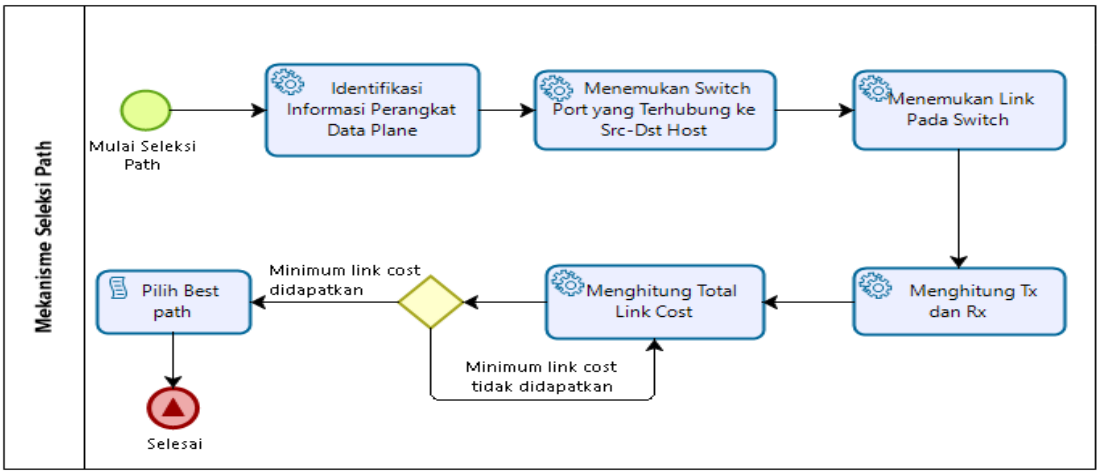
Pada Gambar 2, menjelaskan bagaimana alur pengiriman paket pada *OpenFlow switch* dan kontroler, dimana ketika paket sampai, *switch* akan mencocokkan *rule* pada paket dan yang memiliki prioritas paling tinggi *Action* nya akan dipilih dan yang *match* akan di proses sesuai jalurnya. Jika tidak ada kondisi yang *match*, maka paket akan dikirim ke kontroler dalam bentuk pesan *OpenFlow*. Notifikasi ini disebut *event packet-in*, termasuk sebuah *prefix* dari isi paket yang sampai dengan beberapa tambahan Informasi (cukup dengan memasukkan *header* paket *Layer* 2, 3 dan 4), seperti *port* fisik dimana paket diterima. Dengan menerima notifikasi ini, kontroler akan memutuskan untuk menambahkan *rule* baru ke *flow table* untuk menangani jenis paket seperti ini.

	ETHERNET			IP LAYER					TCP		UDP	
Ingress Port	SA	DA	ETHER TYPE	IP4 SA	IP4 DA	IPV6 SA	IPV6 DA	Protocol	Source Port	Destination Port	Source Port	Destination Port

Gambar 1. *Match field* paket *OpenFlow*.



Gambar 2. Alur diagram *OpenFlow switch* dan kontroler.



Gambar 3. Diagram mekanisme seleksi *path* untuk *load balancing*.

B. Penelitian Terkait

SDN memisahkan *control plane* untuk logika jaringan dan *data plane* untuk fungsi *forwarder*, misalnya, *router* atau *switch*. Akibatnya, informasi manajemen jaringan dan logika jaringan disatukan di seluruh pengontrol SDN (juga dikenal sebagai *control plane*). Pengontrol SDN, yang mengontrol semua fungsi jaringan dengan bantuan protokol *OpenFlow*, memainkan peran utama dalam arsitektur SDN [11].

Menurut [12] peningkatan arus data yang terjadi pada *data center* dapat menyebabkan *delay* yang tinggi dan *throughput* menjadi rendah sehingga dibutuhkan kontrol untuk menangani permasalahan tersebut. Penelitian ini melakukan survei terhadap beberapa mekanisme *congestion control* pada TCP di lingkungan SDN, sehingga didapatkan hasil perbandingan mekanisme *congestion control*.

Penelitian terkait pencegahan kemacetan jaringan pada *data center* juga dilakukan oleh [13]. Penelitian ini melakukan perbandingan teknik pencegahan kemacetan jaringan pada *data center* dengan menggunakan pendekatan replikasi data.

Menurut [14] SDN menggabungkan berbagai jenis teknologi jaringan untuk membuat jaringan lebih skalabel dan dinamis untuk mendukung *server virtual* dan infrastruktur pusat data yang modern, dan pendekatan SDN pada awalnya didefinisikan untuk merancang, membangun dan mengelola jaringan yang memisahkan *control plane* (otak) dan *forwarder plane* (otot) memungkinkan untuk kontrol jaringan langsung yang dapat diprogram dan meng-abstraksi infrastruktur dibawahnya untuk aplikasi dan layanan jaringan.

Penelitian yang dilakukan oleh [15] menggunakan teknik *Round Robin*. Makalah ini tidak menawarkan *load* dan *delay* pada *server*. Pendekatan ini mengasumsikan bahwa semua *server* dalam sistem jaringan tertentu memiliki jumlah permintaan yang sama dan bahwa setiap koneksi memiliki kecepatan yang sama. Mereka menggunakan pengontrol POX untuk menguji *load balancing* berdasarkan statistik *flow*. *Load balancing* dilakukan di sistem mereka untuk mencegah *server* kelebihan beban dengan membagi permintaan koneksi server ke beberapa *server* secara merata. Namun, situasi di dunia nyata sangat berbeda. Hampir setiap koneksi memiliki *bandwidth* dan kecepatan yang berbeda. [16] memperkenalkan strategi *balancing* untuk *load* yang berdasarkan waktu paling sedikit. Dalam studi mereka, permintaan klien akan dikirim oleh *load balancer* ke *server* yang memiliki waktu *delay* paling sedikit, daripada server lain yang memiliki lebih banyak waktu *delay*.

Untuk menghindari beban *link* yang berlebih, [17] menyelidiki *load balancing* tergantung pada kepadatan *traffic*, dengan mengelola penggunaan koneksi dan untuk menyeimbangkan *traffic* antara *link* yang tersedia.

III. METODE YANG DIUSULKAN

Usulan yang kami ajukan adalah dengan metode pemilihan jalur atau *data path selection* untuk mencegah kemacetan atau *congestion*, pemilihan jalur yang kemudian disebut *best path*, *best path* dapat di kategorikan sebagai *path* yang memiliki beban paling rendah dalam hal ini menggunakan algoritma *least loaded path*. Pemilihan *Least Loaded Path* dapat dilakukan pada topology *tree* dan *fat tree*. Topologi *fat tree* memiliki kelebihan dalam skala jaringan besar yang dapat diterapkan pada *cloud data center* maupun *big enterprise*, *fat tree topology* juga dapat mendukung konektivitas antara banyak segmen jaringan yang berbeda, sehingga dapat meningkatkan skalabilitas dan kemanan jaringan. Dalam merancang *load balancing* yang baik, penentuan *best path* atau *shortest path* dibutuhkan, untuk menetapkan jalur mana yang akan dilewati oleh trafik data. Dengan menggunakan *best path* dapat meningkatkan *transfer rate data* pada jaringan dan menghindari *high latency*.

Algoritma *least loaded path* digunakan untuk menentukan jalur terpendek dan *cost* terendah, dengan menghitung statistik jaringan dari *port switch*, penjumlahan nilai data *transmitted* (*Tx*) dan *received* (*Rx*) pada *switchport* akan menentukan jalur atau *path* dengan *cost* terendah. Setelah *best path* ditentukan, pengontrol akan mengirimkan *flow* ke *flow table data plane*, maka *traffic flow* yang diminta akan menggunakan jalur tersebut sebagaimana dinyatakan pada *flow table* atau *rule*. Untuk lebih detail tentang proses mekanisme seleksi path dengan metode least loaded path, akan ditunjukkan pada Gambar 3.

IV. Skenario Uji Coba dan Pengaturan Sistem

Pada penelitian ini, telah dirancang skenario pengujian, berupa topologi *fat tree* seperti yang ditunjukkan pada Gambar 4. dengan *source-destination* (S-D) H1 dan H4 dengan menggunakan kondisi *link occupancy* atau *link load* yang berbeda, *link load low* (25%), *medium* (50%) dan *high* (85%) dari total bandwidth 10 Mbps. Penggunaan *link occupancy* ini dilakukan untuk mensimulasikan kondisi *path congestion* seperti kondisi *real* dan diharapkan dapat melihat kapabilitas mekanisme LLP terhadap *link condition* yang bervariasi, mulai kondisi *low traffic* hingga *high traffic* (*link overloading*). Penggunaan *bandwidth* 10 Mbps diharapkan dapat memberikan hasil evaluasi terhadap kemampuan mekanisme LLP dalam mentransmisikan data dengan ketersediaan *bandwidth* pada suatu jaringan.

Kami menggunakan *floodlight* Versi 1.2 sebagai kontroler terpusat, menggunkan OpenFlow versi 1.1 sebagai protokol komunikasi antara kontroler dan *data plane* atau *switch*, serta menggunakan *mininet* sebagai *emulator* topologi *fat tree* yang berisi *switch* dan *host* yang telah dibuat menggunakan skrip *python*.

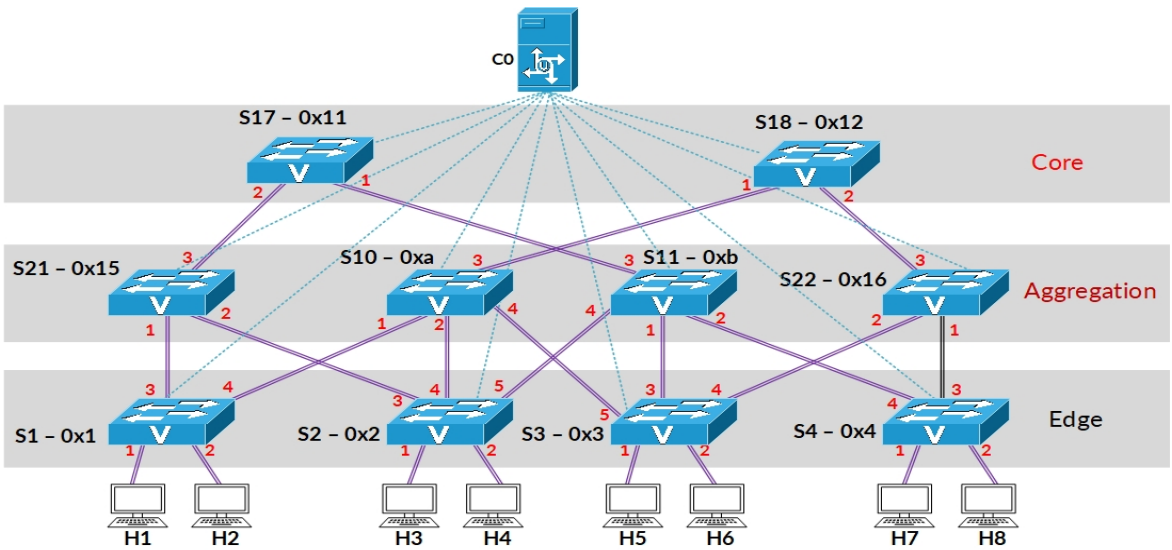
Setelah menjalankan kontroler *floodlight*, topologi *fat tree* dijalankan dengan menambahkan parameter *bandwidth* 10 Mbps, kemudian menambahkan *link load* yang berbeda mulai 25%, 50% dan 85% dari total *bandwidth*, dengan cara mengirimkan paket data dengan *Iperf* dari H1 ke H4, sebelum dilakukan pengukuran antara S-D H1 dan H4, akan dilakukan simulasi kemacetan pada *path* atau jalur dengan mengirimkan paket data sebesar 64 *Bytes* dari H1 ke H3 dengan protokol ICMP, hal ini dilakukan untuk mendapatkan statistik *port* serta mengetahui perubahan transmisi data sebelum dan sesudah proses *seleksi* path.

Untuk menentukan jalur terpendek dan *cost* terendah, digunakan algoritma *Least Loaded Path* (LLP) dengan melakukan penjumlahan Tx (*Transmission rate data*) dan Rx (*Receiving rate data*) dengan persamaan (1) pada *switchport* yang telah terhubung dengan H1, H3 dan H4. Setelah jalur terbaik ditentukan, pengontrol akan mengirimkan *flow* ke tabel *flow data plane*, maka *traffic flow* yang diminta akan menggunakan jalur tersebut sebagaimana dinyatakan pada tabel *flow* atau pada *rule*.

$$C_m = TX_p + RX_p \quad (1)$$

dengan:

C_m = *Cost* minimum dari penjumlahan data pada *switchport*
TX_p = Data yang ditransmisikan pada *switchport*
RX_p = Data yang diterima pada *switchport*



Gambar 4. Topology *fat-tree*.

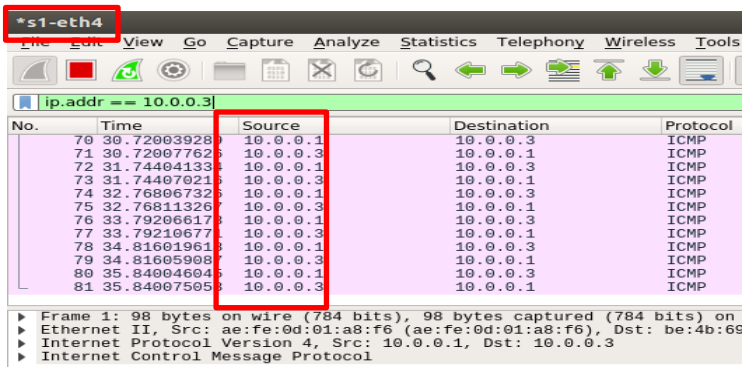
V. HASIL DAN PEMBAHASAN

Dari hasil uji coba yang telah dilakukan, terdapat penurunan yg signifikan pada *latency*, serta peningkatan *transfer rate data* dan *throughput* setelah dilakukan proses seleksi *path* dengan *least loaded path*. Hal ini menunjukkan proses *seleksi path* telah menunjukkan peningkatan performa pada jaringan SDN *data plane*. Selain itu, hasil ini menunjukkan ketika *path* yang dipilih menjadi *best path*, maka kontroler akan mengirimkan *flow* ke *flow table* kepada *switch* atau *data plane* yang terlibat dan sudah terpilih menjadi *best path*, sehingga setiap *host* yang akan mengirimkan paket data, akan mengikuti rule yang sudah tercatat pada *flow table data plane*. Hal ini menjadikan mekanisme ini menjadi reaktif, dimana ketika terdapat *path* yang baru, kontroler baru mengirimkan *flow* ke *data plane*.

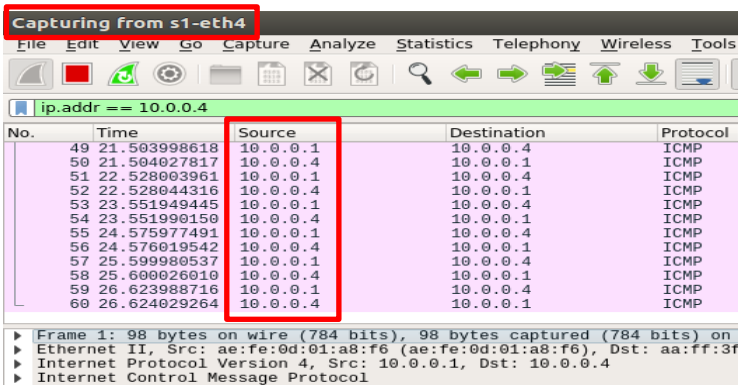
Skenario pengujian S-D H1-H4 dengan *bandwidth* 10 Mbps menggunakan topologi jaringan *fat tree* seperti yang ditunjukkan pada Gambar 4. Uji coba ini dilakukan untuk melihat apakah mekanisme LLP dapat bekerja dengan baik dan membandingkan hasil kinerja nya dengan algoritma LB *default* pada *floodlight* untuk memastikan semua *host* terhubung, kami menggunakan perintah "*pingall*" pada *mininet*. Pada saat dilakukan pengujian dengan kondisi *load network; low* (25%), *medium* (50%) dan *high* (85%) dari total *bandwidth* 10 Mbps, didapatkan hasil *latency* dengan mekanisme LLP lebih rendah dan peningkatan *transfer rate* data, dibandingkan menggunakan *path* dengan algoritma *default*. Ketika menggunakan default LB *floodlight*, *best path* atau *route* yang digunakan oleh H1 untuk melakukan transfer paket ke H3 dan H4 adalah [S1:4-S10:2-S2:4], paket data yang dikirim ke dua *host* yang berbeda menggunakan *path* yang sama, sehingga menyebabkan *latency* menjadi tinggi. Untuk membuktikannya dengan menggunakan *packet capture wireshark*, yang dapat dilihat pada Gambar 5 dan Gambar 6.

Pada Gambar 5 dan 6, dapat dilihat pengiriman paket data baik dari S-D H1-H3 dan H1-H4, sama-sama melalui S1 port 4. Hal ini menunjukkan *default best path* nya adalah melalui [S1:4-S10:2-S2:4]. Sebelum dilakukan mekanisme seleksi path dengan LLP, dilakukan pengukuran dari H1 ke H4, dengan menambahkan parameter-parameter yang telah dijelaskan sebelumnya. Setelah dilakukan mekanisme seleksi *path*, Gambar 7 menunjukkan pengiriman paket data dari H1-H3 tidak melalui S1 port 3, namun tetap melewati *path* sebelumnya, yaitu [S1:4-S10:2-S2:4].

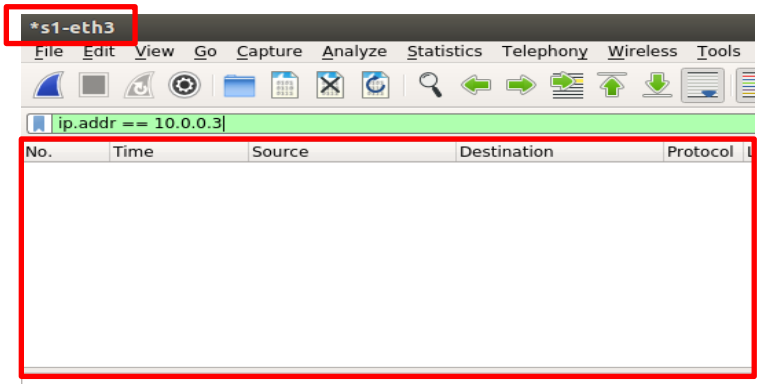
Sementara Gambar 8 menuunjukkan, setelah dilakukan mekanisme seleksi *path* dengan LLP, maka pengiriman paket data dari H1 ke H4 menggunakan *path* yang baru [S1:3-S21:2-S2:3], dimana *path* tersebut dinilai memiliki *cost* terendah sehingga dipilih sebagai *best path*. Proses *capture* paket dan pengukuran latensi pada skenario H1-H4 ini menunjukkan bahwa mekanisme LLP telah berhasil dalam melakukan *load balancing* pada *path* yang telah ditentukan.



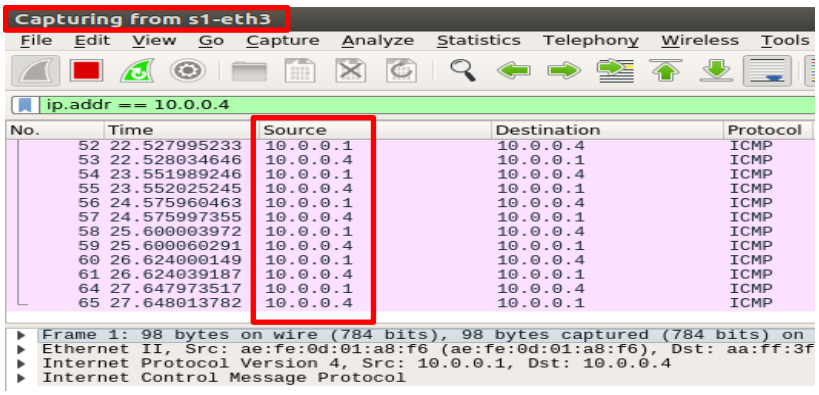
Gambar 5. *Capture* paket data H1-H3 pada S1 *port* 4 tanpa seleksi *path*.



Gambar 6. *Capture* paket data H1-H4 pada S1 *port* 4 tanpa seleksi *path*.



Gambar 7. *Capture* paket data H1-H3 pada S1 *port* 3 dengan seleksi *path.*



Gambar 8. *Capture* paket data H1-H4 pada S1 *port* 3 dengan seleksi *path.*

Untuk perbandingan hasil *latency* pada kondisi *network load low*, *medium*, dan *high* dapat dilihat pada Tabel I dan Tabel II serta grafik perbandingan latensi menggunakan default LB dan mekanisme LLP ditunjukkan pada Gambar 9. Berdasarkan Gambar 9, dapat dilihat bahwa hasil latensi dengan mekanisme LLP mengalami penurunan dibandingkan dengan *default path*, penurunan latensi juga terjadi pada *link load* yang lebih tinggi, hal ini disebabkan setelah menggunakan mekanisme LLP, maka *path* yang baru akan dipilih untuk mengirimkan paket data dari H1 ke H4 yaitu, melalui path [S1:3-S21:2-S2:3], sehingga waktu yang dibutuhkan untuk mengirimkan paket data lebih cepat. Sementara pengiriman paket data dari H1 ke H3 tetap menggunakan *path* [S1:4-S10:2-S2:4], ini dapat dibuktikan dengan melakukan capture paket data dari H1 ke H3 pada S1 *port* 3, serta *capture* paket data dari H1 ke H4 pada S1 *port* 3 seperti yang ditunjukkan pada Gambar 7 dan Gambar 8.

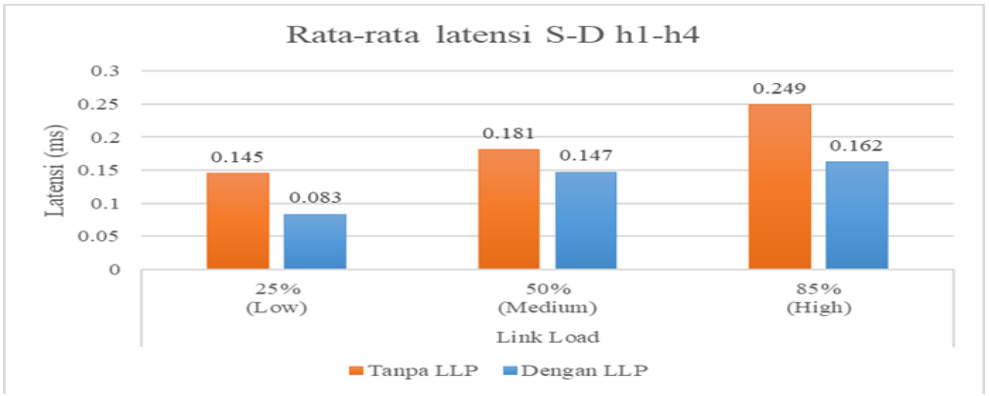
Pada Gambar 10, menunjukkan grafik perbandingan maksimum latensi antara penggunaan default LB dan mekanisme LLP, menggunakan parameter *link load* yang berbeda, pengujian dilakukan sebanyak 100 kali pengiriman paket data dengan protokol ICMP. Rata-rata latensi menggunakan default LB (tanpa LLP) pada kondisi *link load* 25% adalah 0,145 ms, sementara menggunakan LLP latensinya turun menjadi 0,072 ms, hal ini disebabkan karena best path baru yang hanya digunakan untuk mengirimkan paket data dari h1 ke h4, sementara paket data dari h1 ke h3 tetap menggunakan jalur yang lama, atau dapat dikatakan pendistribusian paket pada *dual path* telah dilakukan dengan menggunakan mekanisme seleksi *path.*

TABEL I
SKENARIO I: RATA-RATA LATENSI S-D H1-H4 TANPA LLP.

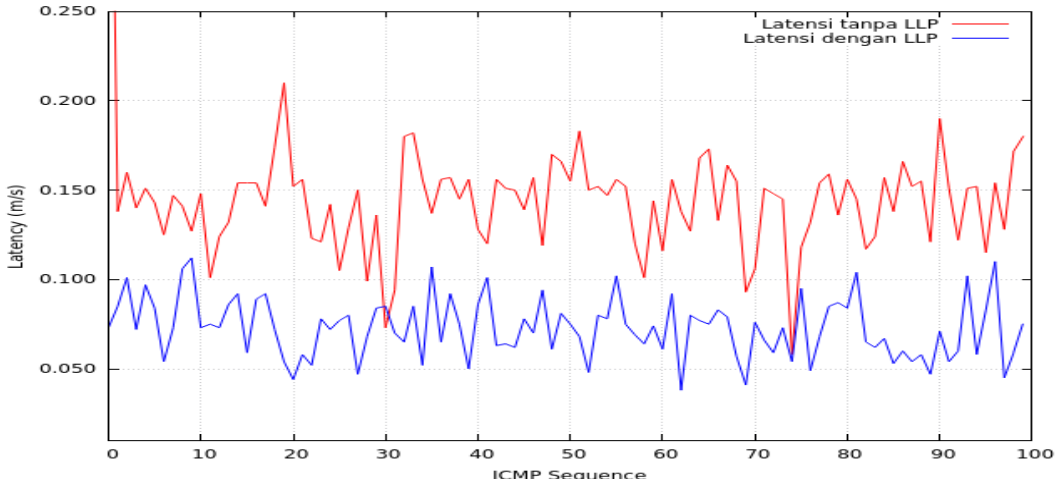
Rata-rata Latensi tanpa seleksi *path* LLP (ms)		
25% (Low)	50% (Medium)	85% (High)
0.145	0.181	0.249

TABEL II
SKENARIO I: RATA-RATA LATENSI S-D H1-H4 DENGAN LLP.

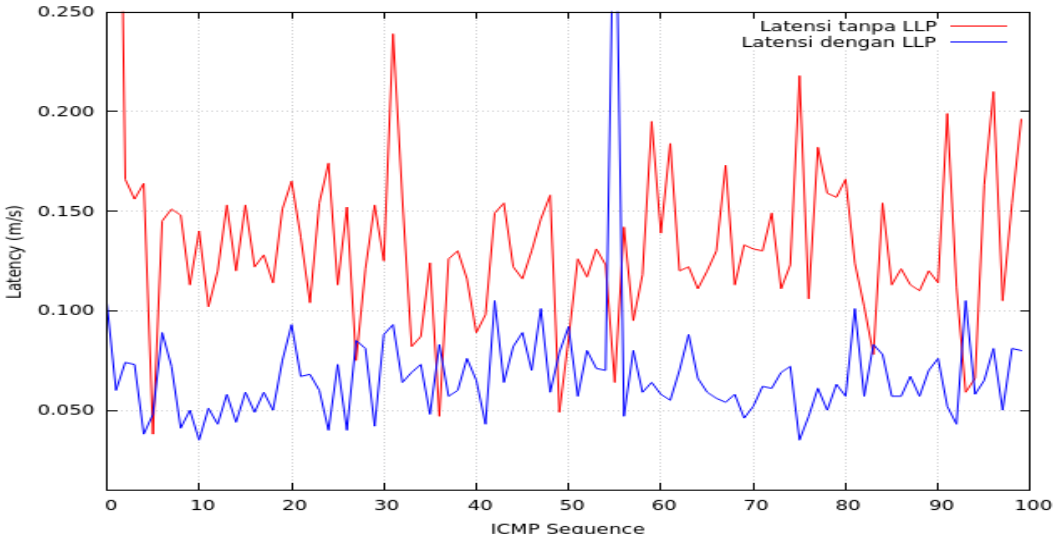
Rata-rata Latensi dengan seleksi *path* LLP (ms)		
25% (Low)	50% (Medium)	85% (High)
0.083	0.147	0.162



Gambar 9. Grafik skenario H1-H4 10 Mbps.



Gambar 10. Perbandingan maksimum latensi H1-H4 10 Mbps *link load* (25%).

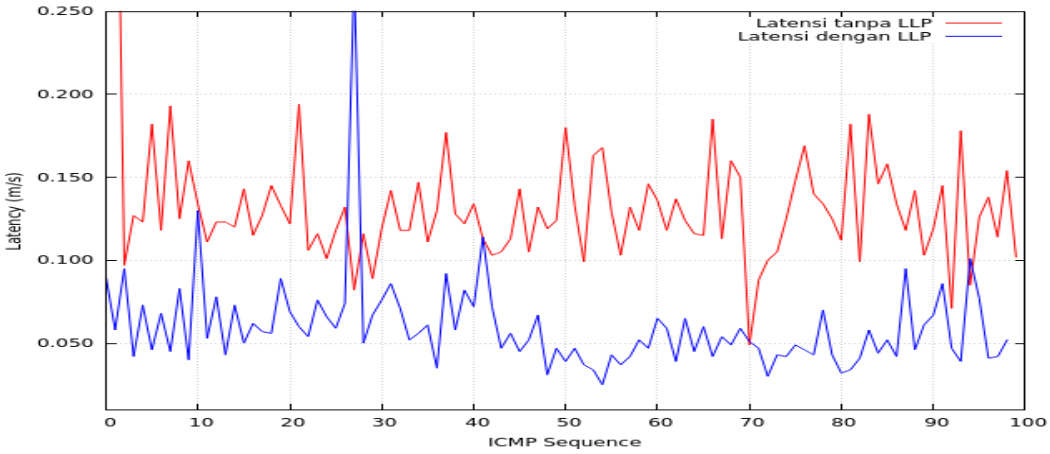


Gambar 11. Perbandingan maksimum latensi H1-H4 10 Mbps *link load* (50%).

Hasil penurunan latensi juga ditunjukkan pada Gambar 11, dimana sebelum menggunakan LLP rata-rata latensinya adalah 0,181 ms, sementara setelah LLP menjadi 0,068 ms. Sama seperti pengukuran pada link load 25%, pada link load 50%, penurunan latensi terjadi karena pendistribusian paket menggunakan *dual path* dari h1 ke h3 dan h4 telah berhasil dilakukan, sehingga pengiriman paket data dapat dilakukan lebih cepat.
Sementara pada Gambar 12 menunjukkan, hasil penurunan latensi setelah dilakukan LLP dengan *link load* 85%, dimana pada saat menggunakan *default* LB rata-rata latensinya adalah 0,249 ms, sedangkan setelah menggunakan LLP latensi menjadi 0,059 ms. Dari ketiga pengukuran latensi dengan *link load* yang berbeda, dapat dilihat penurunan latensi selalu terjadi, hal ini membuktikan mekanisme seleksi *path* dengan LLP telah berjalan dengan baik dan mampu meningkatkan waktu pengiriman paket data. Untuk hasil *transfer rate* data dan *throughput*, di-

tunjukkan pada Tabel III dengan menggunakan *link load* yang berbeda, yaitu 25%, 50% dan 85% dari total 10 Mbps *bandwidth.*

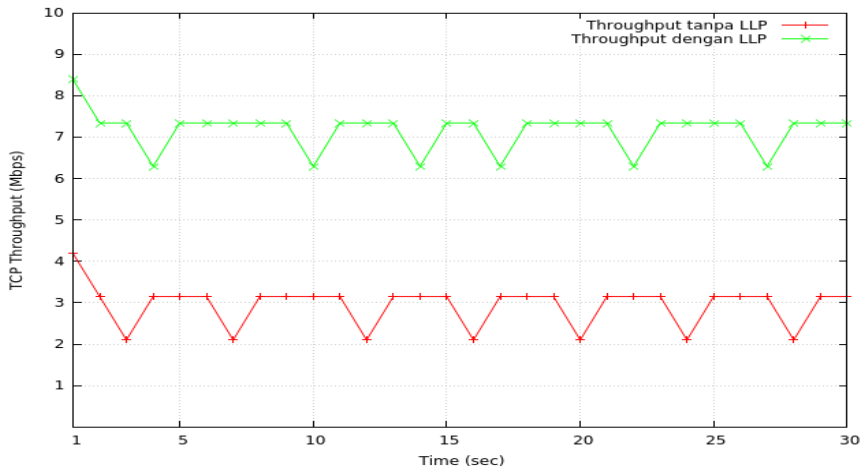
Pada Tabel III, merupakan hasil pengujian selama 30 detik dengan *iperf* menggunakan default TCP *window size* 85,3 Kbyte, *transfer rate* data dan t*hroughput* (*bandwidth*) menunjukkan peningkatan selalu terjadi setelah menggunakan mekanisme seleksi *path* dengan LLP, pada kondisi *link load low*, terjadi peningkatan *transfer rate* data sebesar 25,2 *Megabytes* atau 98% dan *throughputnya* meningkat sebesar 4,24 Mbps atau 59%. Pada kondisi link load *medium,* terjadi peningkatan *transfer rate* data sebesar 8,3 *Megabytes* atau 49% dan *throughputnya* meningkat sebesar 2,32 Mbps atau 50%. Sedangkan pada kondisi *link load high,* terjadi peningkatan *transfer rate* data sebesar 2,1 *Megabytes* atau 46% dan *throughputnya* meningkat sebesar 0,66 Mbps atau 52%. Hal ini dapat terjadi karena, setelah dilakukan mekanisme seleksi *path* dengan LLP, maka pengiriman paket data tidak hanya menggunakan satu *path* saja, namun menggunakan *dual path*, sehingga pendistribusian pengiriman data lebih merata dan seimbang. Namun ketika kondisi *load* semakin bertambah, *transfer rate* data dan *throughput* juga menjadi turun, ini menunjukkan kapasitas *bandwidth* sudah sesuai dengan skenario pengujian dan pemanfaatan *bandwidth* menjadi maksimal. Untuk hasil perbandingan *throughput* S-D H1-H4 dengan menggunakan *link load* yang berbeda pada alokasi *bandwidth* 10 Mbps dapat dilihat pada Gambar 11, Gambar 12, dan Gambar 13.



Gambar 12. Perbandingan maksimum latensi H1-H4 10 Mbps *link load* (85%).

TABEL III
SKENARIO I: TRANSFER RATE DAN THROUGHPUT H1-H4 10 MB.

	Default LB						Dengan LLP					
	Transfer Rate Data (Kbytes)			*Throughput* (Kbps-Mbps)			*Transfer Rate* Data (Kbytes)			*Throughput* (Kbps-Mbps)		
Interval (sec)	25% Load	50% Load	85% Load	25% Load	50% Load	85% Load	25% Load	50% Load	85% Load	25% Load	50% Load	85% Load
0-30	0,6 MB	8,6 MB	2,5 MB	2,9 Mbps	2,35 Mbps	633 Kbps	25,8 MB	16,9 MB	4,6 MB	7,14 Mbps	4,67 Mbps	1,26 Mbps



Gambar 13. Perbandingan *throughput* H1-H4 10 Mbps *link load* (25%).

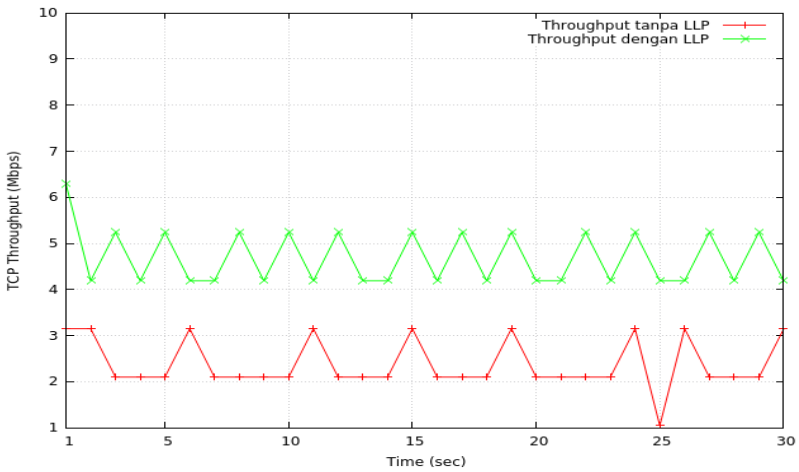
Gambar 13 menunjukkan, grafik perbandingan *throughput* yang menggunakan *default* LB dan menggunakan LLP dengan kondisi *link load* 25% dari total *bandwidth* 10 Mbps atau 2,5 Mbps. Grafik tersebut menunjukkan terjadi kenaikan *throughput* setelah menggunakan LLP, dimana terjadi kenaikan rata-rata 3 Mbps menjadi 7 Mbps setelah diuji selama 30 detik dengan *default* TCP *window size default* 85,3 *Kbyte*. Hal ini menunjukkan mekanisme LLP dapat meningkatkan *throughput*, karena pada saat pengiriman paket data dari H1 ke H3 dan ke H4 menggunakan *dual path*, sedangkan pada default LB hanya menggunakan satu *path* saja.

Pada Gambar 14, menunjukaan hasil peningkatan dengan kondisi *link load* 50%, rata-rata peningkatan *throughput* nya adalah dari 2-3 Mbps menjadi 4-5 Mbps, peningkatan yang terjadi tidak sebesar pada kondisi *link load* 50%, hal ini disebabkan kondisi *load path* yang semakin bertambah, sehingga pengiriman paket hanya memanfaatkan 50% sisa *bandwidth*. Namun hal ini sudah menunjukkan peningkatan dalam pengiriman paket data menggunakan mekanisme LLP.

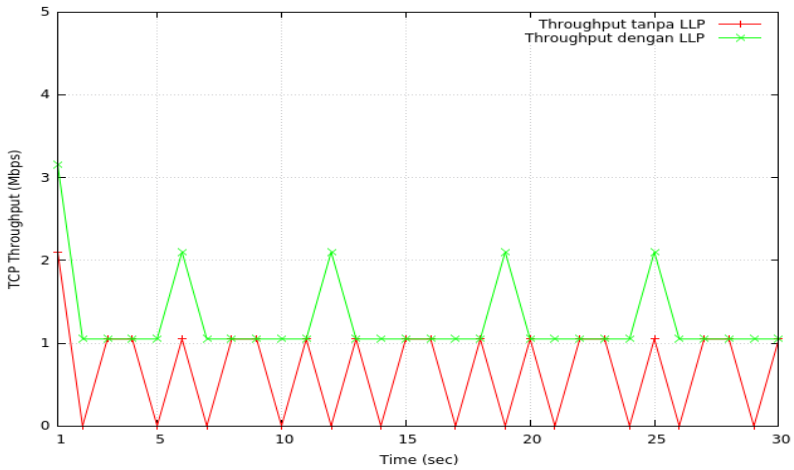
Pada Gambar 15, juga menunjukkan hasil peningkatan *throughput* pada kondisi *link load* 85%, yaitu rata-rata peningkatan dari 0-1 Mbps menjadi 1-2 Mbps. Karena load path yang semakin bertambah, maka pengiriman paket data dari h1 ke h3 dan h4 juga semakin kecil. Hal ini ditandai dengan *throughput* yang semakin menurun dari pengujian menggunakan *link load* 25% dan 50%. Namun peningkatan *throughput* tetap terjadi setelah dilakukan mekanisme seleksi path dengan LLP.

Sebagai pengujian untuk melihat apakah kondisi *path* atau jalur pada jaringan telah seimbang, dilakukan pengujian menggunakan JPerf, dengan mengirimkan paket data ke H3 dan H4 dari *source* yang sama yaitu H1. Hasilnya menunjukkan baik path [S1:3-S21:2-S2:3] dan [S1:4-S10:2-S2:4] dapat mengirimkan paket data dengan jumlah yang hampir sama, hal ini dapat membuktikan bahwa kondisi jaringan sudah seimbang terutama pada *data plane*.

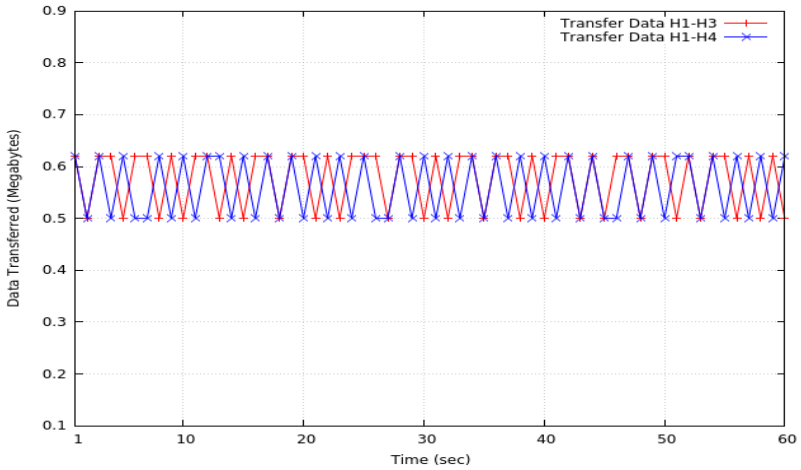
Dari hasil pengujian dengan *JPerf*, selama 60 detik, TCP *Window Size* 56 *Kbyte* pada *bandwidth* 10 Mbps, menunjukkan keseimbangan pengiriman data dari H1 ke H3 dan dari H1 ke H4, transmisi data dilakukan secara bersamaan menunjukkan jumlah data yang ditransmisikan cukup stabil dan seimbang pada range 0,50 *Megabytes* hingga 0,62 *Megabytes* seperti yang dtampilkan pada Gambar 16.



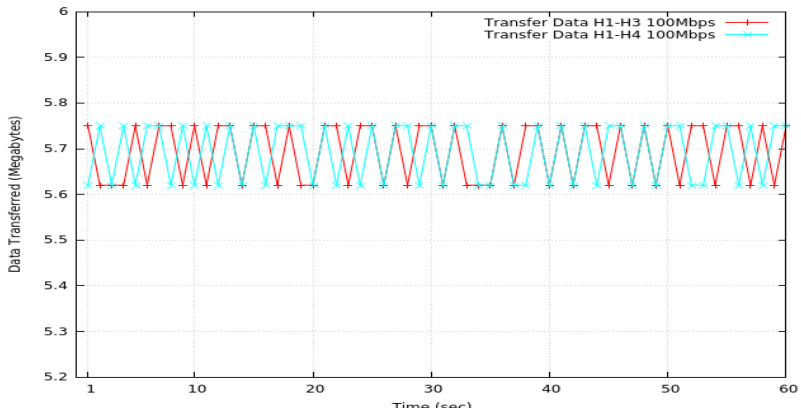
Gambar 14. Perbandingan *throughput* H1-H4 10 Mbps *link load* (50%).



Gambar 15. Perbandingan *throughput* H1-H4 10 Mbps *link load* (85%).



Gambar 16. Grafik transfer data H1-H3-H4 10 Mbps



Gambar 17. Grafik transfer data H1-H3-H4 100 Mbps.

Sementara hasil pengiriman data yang stabil dan seimbang ditunjukkan pada Gambar 17. Hasil pengujian dengan *JPerf*, selama 60 detik, TCP *Window Size* 56 *Kbyte* menggunakan *bandwidth* 100 Mbps, untuk pengiriman data pada range 5,62 *Megabytes* hingga 5,75 *Megabytes*.

VI. KESIMPULAN DAN SARAN

Setelah melakukan pengujian dengan skenario S-D H1-H4 10 Mbps menggunakan *link load* yang bervariasi, dengan menerapkan mekanisme seleksi *path* menggunakan metode *least loaded path* (LLP) secara dinamis, dapat dilihat *load balancer* mendefinisikan beberapa jalur untuk jalur terbaik dan merutekan ulang jalur yang ditentukan secara merata, sehingga pengiriman paket dapat dilakukan melalui dua *path*. Perbandingan kinerja antara *load balancing* terintegrasi dan mekanisme LLP telah dilakukan. *Load balancer* yang diusulkan memiliki kinerja yang lebih baik yang ditunjukkan oleh peningkatan *bandwidth*, kecepatan transfer data dan mengurangi latensi. Berdasarkan uji coba dari skenario H1-H4 dengan *link load* 25%, 50% dan 85% *bandwidth* 10 Mbps, mengalami penurunan rata-rata *latency* 50% hingga 79%, dan peningkatan *transfer rate* data sebesar 46% hingga 98% serta peningkatan *throughput* sebesar 50% hingga 59%, setelah dilakukan proses mekanisme seleksi *path.* Kami mengamati bahwa peningkatan akan lebih efisien untuk *load balancing* jika ada beberapa jalur alternatif yang tersedia. Hal ini juga membuat *load balancer* bekerja lebih cepat ketika jaringan memiliki *link* atau *path* alternatif untuk dipilih sebagai jalur terbaik. Dari beberapa pengujian tersebut, dapat disimpulkan bahwa mekanisme seleksi path dengan LLP telah berhasil menunjukkan kinerja yang baik dalam menangani distribusi pengiriman paket data pada dua path yang berbeda, pada kondisi link load yang rendah maupun tinggi. Hal ini menunnjukkan mekanisme LLP dapat mencapai load balancing pada jaringan SDN *data plane*. Kami dapat menyarankan mekanisme ini untuk diterapkan pada skala jaringan apa pun (skala kecil atau jaringan skala besar), meskipun pada jaringan skala besar performanya dapat lebih baik. Penerapan mekanisme ini juga akan membuat jaringan lebih efisien dan lebih terukur. Kami sadar bahwa penelitian ini memiliki potensi pengembangan lain, kami dapat mengusulkan topologi lain dengan ratusan *switch* dan ribuan *host* untuk menerapkan *load balancer* sebagai

penelitian kedepannya, kita akan tahu apakah *load balancing* cocok untuk jaringan dengan kepadatan yang tinggi. Oleh karena itu kami juga dapat mensimulasi *load balancing* di jaringan *traffic* yang tinggi, untuk mengamati kinerja dan kompatibilitas jaringan SDN.

DAFTAR PUSTAKA

[1] Y. R. Chiang, C. H. Ke, Y. S. Yu, Y. S. Chen, dan C. J. Pan, "A multipath transmission scheme for the improvement of throughput over SDN," dalam *Proc. IEEE Int. Conf. Appl. Syst. Innov. Appl. Syst. Innov. Mod. Technol.*, hal. 1247–1250, 2017.

[2] J. R. Jiang, H. W. Huang, J. H. Liao, dan S. Y. Chen, "Extending Dijkstra's shortest path algorithm for software defined networking," in *Proc. Asia-Pacific Netw. Oper. Manag. Symp.*, 2014.

[3] S. Wang, J. Luo, B. K.-B. Tong, dan W. S. Wong, "Randomized Load-balanced Routing for Fat-tree Networks," *arXiv*, hal. 1–13, 2017. Tersedia: http://arxiv.org/abs/1708.09135.

[4] Y. Li dan D. Pan, "OpenFlow based load balancing for fat-tree networks with multipath support," dalam *Proc. IEEE Int. Conf. Commun.*, hal. 1–5, 2013.

[5] C. X. Cui dan Y. Bin Xu, "Research on load balance method in SDN," *Int. J. Grid Distrib. Comput.*, vol. 9, no. 1, hal. 25–36, 2016.

[6] A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh, dan A. Rezaee, "Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature," *IEEE Access*, vol. 6, hal. 14159–14178, 2018.

[7] D. Mithbavkar, H. Joshi, H. Kotak, D. Gajjar, dan L. Perigo, "Round robin load balancer using software defined networking (SDN)," *Capstone Team Res. Proj.*, hal. 1–9, 2016.

[8] G. Tiwari, A. Rai, V. Deeban Chakaravarthy, and R. Kadikar, "A survey paper on dynamic load balancing in software defined networking," *Int. J. Recent Technol. Eng.*, vol. 7, no. 6, hal. 1996–1998, 2019.

[9] N. Mckeown, T. Anderson, L. Peterson, J. Rexford, S. Shenker, and S. Louis, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, hal. 69–74, 2008.

[10] M. P. Fernandez, "Comparing openflow controller paradigms scalability : reactive and proactive," dalam *Proc. IEEE International Conference on Advanced Information Networking and Applications*, 2013.

[11] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, dan S. Uhlig, "Software-defined networking: a comprehensive survey," dalam *Proc. IEEE*, vol. 103, no. 1, hal. 14–76, 2015.

[12] F. H. Saputra dan R. M. Ijtihadie, "Survei mekanisme congestion kontrol pada transmission control protocol di software defined network," *JUTI J. Ilm. Teknol. Inf.*, vol. 16, no. 1, hal. 1, 2018.

[13] E. Auparay dan R. M. Ijtihadie, "Replikasi data menggunakan detection controller module untuk mencegah congestion di data center," *JUTI J. Ilm. Teknol. Inf.*, vol. 16, no. 1, hal. 10, 2018.

[14] N. Feamster, J. Rexford, dan E. Zegura, "The road to SDN: an intellectual history of programmable networks," *Comput. Commun. Rev.*, vol. 44, no. 2, hal. 87–98, 2014.

[15] K. Kaur, S. Kaur, dan V. Gupta, "Flow statistics based load balancing in openflow," dalam *Proc. Int. Conf. Adv. Comput. Commun. Informatics,* hal. 378–381, 2016.

[16] K. Kaur, S. Kaur, dan V. Gupta, "Least time based weighted load balancing using software defined networking," dalam *Proc. International Conference on Advances in Computing and Data Sciences*, hal. 309-314, 2016.

[17] S. Bhandarkar dan K. A. Khan, "Load balancing in software-defined network (SDN) based on traffic volume," *Adv. Comput. Sci. Inf. Technol.*, vol. 2, no. 7, hal. 72–76, 2015.