REPOSITOR, Vol. 2, No. 12, Desember 2020, Pp. 1727-1738
ISSN : 2714-7975
E-ISSN : 2716-1382

Analisa Performansi Controller Pada Arsitektur Jaringan Software Defined Network (SDN)

Agung Dwi Rahmawan*[1], Syaifuddin[2], Diah Risqiwati[3]
[1,2,3] Universitas Muhammadiyah Malang
e-mail: agung_dr4@webmail.umm.ac.id[*1], saifuddin@umm.ac.id[2], rizqiwati@umm.ac.id[3]

Abstrak

Software Defined Network (SDN) merupakan sebuah konsep pendekatan baru dalam jaringan untuk mendesain, membangun serta mengelola suatu jaringan komputer. Konsep ini melakukan pemisahan terhadap Data Plane dan Control Plane. Dalam konsep SDN ini terdapat suatu komponen penting yang bertanggung jawab terhadap segala aturan dalam pengelolaan dan pendistribusian informasi terhadap seluruh perangkat jaringan yaitu Controller. Karena peran Controller yang penting maka performa dari Controller perlu diuji sehingga dapat mengetahui kemampuan dari Controller yang digunakan. Dalam penelitian ini dilakukan perbandingan analisis nilai Quality of Services (QoS) terhadap implementasi SDN menggunakan Controller Floodlight dan Ryu dengan menjalankan topologi linear dan mesh dalam jumlah Switch yang beragam mulai dari 4, 8, 12 dan 16 Switch. Selama pengujian berlangsung dari node sumber ke node tujuan yang sama juga dialiri variasi background traffic mulai dari 50 hingga 200 Mbps. Hasil yang didapatkan yaitu Controller Ryu memiliki nilai QoS yang lebih baik dari floodlight pada semua topologi yang diujikan, nilai latency dan jitter pada floodlight lebih tinggi dari ryu serta cenderung meningkat pada traffic 100 Mbps Pada throughput, ryu memiliki nilai lebih tinggi dengan kisaran 856-933 Kbps. *Sedangkan pada packet loss floodlight lebih tinggi sementara ryu hanya memiliki rata-rata packet loss sebesar 0,5%. Namun pada pengujian hanya pada jumlah switch, floodlight menjamin dalam tingkat respons serta pengelolaan data yang besar di dalam arsitektur jaringan SDN.*

Kata kunci: *Software Defined Network, Floodlight, Ryu, Mininet*

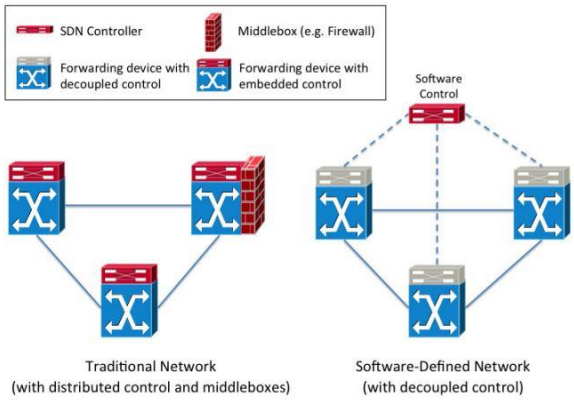
Abstract

Software Defined Network (SDN) is a concept of a new approach in networking to design, build and manage a computer network. This concept separates the Data Plane and Control Plane. In this SDN concept there is an important component that is responsible for all rules in the management and distribution of information to all network devices that is Controller. Due to the important Controller role then the performance of the Controller needs to be tested so as to know the ability of the Controller to use. In this study, a comparison of Quality of Service (QoS) value analysis on SDN implementation using Floodlight and Ryu Controller by running linear and mesh topology in varying number of Switches ranging from 4, 8, 12 and 16 Switch. During the test from the source node to the same destination node is also varies background traffic ranging from 50 to 200 Mbps. The result is that Controller Ryu has better QoS value than floodlight on all tested topologies, the latency and jitter values on the floodlight are higher than ryu and tend to increase on 100 Mbps traffic. Throughput On ryu. have a higher value with the range of 856-933 Kbps. While the packet loss floodlight higher while ryu only have an average packet loss of 0.5%. But on testing only on the number of switches, the floodlight guarantees great response rates and data management within the SDN network architecture.

Keywords*: Software Defined Network, Floodlight, Ryu, Mininet*

1. Pendahuluan

Perkembangan teknologi informasi saat ini sangat pesat. Hal ini juga berpengaruh pada layanan internet yang berkembang dengan berbagai kompleksitas, desain, manajemen serta operasionalnya. Permasalahan terjadi ketika ada berbagai *hardware* dengan jenis serta protokol yang berbeda terdapat dalam sebuah jaringan. Dalam beberapa tahun terakhir konsep *Software Defined Network* (SDN) menjadi pilihan oleh beberapa perusahaan besar untuk memproduksi dan memanfaatkan hardware mereka sendiri demi memudahkan manajemen dan pengembangan pada jaringan mereka [1].

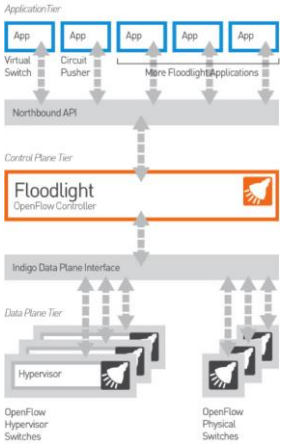


Gambar 1 Perbandingan Jaringan Tradisional Dengan ***[2]***

Dari **Gambar 1** terlihat perbandingan antara jaringan tradisional dengan Software Defined Network (SDN). Pada jaringan tradisional *control plane* merupakan struktur yang terdistribusi, dimana setiap perangkat *networking* masing-masing memiliki *control plane* sehingga dapat melakukan distribusi data secara tersendiri tiap perangkat yang ada. Sedangkan *control plane* pada jaringan SDN memliki struktur yang terpusat, dimana perangkat networking membutuhkan sebuah controller dalam memberikan keputusan untuk melakukan distribusi data.

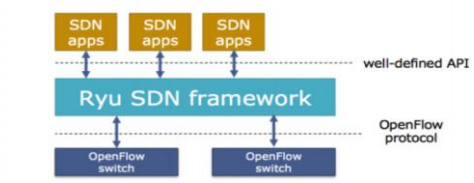
SDN merupakan sebuah konsep baru pada pendekatan dalam merancang, menyusun serta mengelola jaringan komputer. Konsep ini berkenaan dengan arsitektur perangkat jaringan seperti router, packet switch, switch LAN dan lain sebagainya. Sebuah perangkat jaringan memiliki dua bagian yaitu data plane dan control plane. Konsep dari SDN ini menerapkan pemisahan antara data plane dan control plane, yang mana data plane tetap berada pada perangkat jaringan, sedangkan control plane terdapat dalam sebuah *entity* terpisah yang dinamakan sebagai controller.*Controller* merupakan perangkat lunak pengendali pada SDN yang mengelola *OpenFlow*. Controller bertanggung jawab dalam menentukan bagaimana menangani paket dan mengelola tabel *flow* dengan menambahkan atau menghapus isi tabel *flow* melalui *secure channel*. Pada dasarnya sebuah *controller* memusatkan kecerdasan jaringan, sedangkan jaringan mempertahankan data plane yang di distribusikan *Switch OpenFlow* dan *Router*. Oleh karena itu sebuah controller menyediakan antarmuka untuk mendesain, membangun dan mengelola tabel *flow* pada *switch* ini [3].

Beberapa contoh dari *controller* yaitu *floodlight* dan *ryu*. *Floodlight* Merupakan *OpenFlow controller* berlisensi *Apache* serta berbasis JAVA. Controller ini merupakan pengembangan dari Beacon controller yang sebelumnya dibuat oleh David Ericsson di Stanford University. Controller floodlight juga didukung oleh komunitas pengembang termasuk beberapa insinyur dari Big *Switch Networks*. *Floodlight* dirancang untuk bekerja dengan meningkatnya jumlah *switch*, *router*, *virtual switch* serta jalur akses yang mendukung standar *OpenFlow* [4]. Pada **Gambar 2** merupakan *framework* pada *floodlight*.



Gambar 2 Floodlight Framework ***[4]***

Sedangkan *ryu* merupakan salah satu *controller* yang fungsinya memusatkan kontrol jaringan untuk mengatur ribuan perangkat jaringan. *Ryu* adalah salah satu *OpenFlow* controller yang bersifat *free*. *Ryu* berasal dari bahasa jepang yang berarti "*flow*". *Ryu controller* berbasis bahasa *Python*. *Ryu* dirancang untuk meningkatkan kelincahan jaringan dengan membuat mudah dalam pengelolaan serta penyesuaian penanganan *traffic* data. *Ryu* menyediakan komponen perangkat lunak dengan API yang didefinisikan dengan baik yang memudahkan bagi pengembang untuk menciptakan manajemen jaringan baru serta pengendalian aplikasi. [5]. Pada **Gambar 3** merupakan *framework* pada *ryu.*



Gambar 3 Ryu Framework ***[5]***

Pada penelitian ini *controller* akan dijalankan pada *emulator mininet* yang merupakan sebuah emulator jaringan yang membuat *virtual hosts, switches, controllers,* dan *links*. *Host Mininet* menjalankan perangkat lunak jaringan, serta *switch* yang mendukung *OpenFlow* untuk *routing* yang sangat fleksibel dan *Software Defined Networking. Mininet* mendukung penelitian, pengembangan, pembelajaran, *prototyping*, pengujian, *debugging*, dan lainnya yang dapat memanfaatkan jaringan eksperimental yang lengkap pada PC. *Mininet* menyediakan cara yang mudah dalam memperoleh perilaku sistem yang benar dan dapat digunakan untuk bereksperimen dengan topologi [6].

Terdapat sepuluh hal yang dapat dipertimbangkan dalam memilih jenis *controller* pada SDN [7], salah satu diantaranya adalah performa dari controller tersebut. Pada penelitian yang dilakukan oleh Rikie Kartadie, *controller Floodlight* memiliki performa lebih baik daripada Opendaylight dalam jumlah switch yang besar [8]. Kemudian dalam penelitian Sawung Murdha A, Floodlight juga lebih menjamin pengelolaan data dalam jumlah besar dan pengaturan aliran data yang lebih tinggi dibanding dengan POX [9]. Sedangkan pada penelitian yang dilakukan oleh Yimeng Zhao dkk, controller ryu memiliki performa yang lebih baik dibanding controller lain dalam penelitiannya dari segi pengelolaan switch baru terhadap *switch* yang lain secara adil [10].

Oleh karena itu, dalam penelitian ini akan dilakukan pengujian terhadap performansi dari controller *Floodlight* dan Ryu untuk mengetahui performa dari kedua *controller* tersebut. Penelitian ini mensimulasikan jaringan SDN dengan menggunakan *emulator mininet*. Dari kedua *controller*, akan di uji performansinya yang diukur dari *latency*, *throughput*, *jitter* dan *packet loss*. Pengujian akan dilakukan beberapa kali agar mendapatkan hasil yang maksimal.

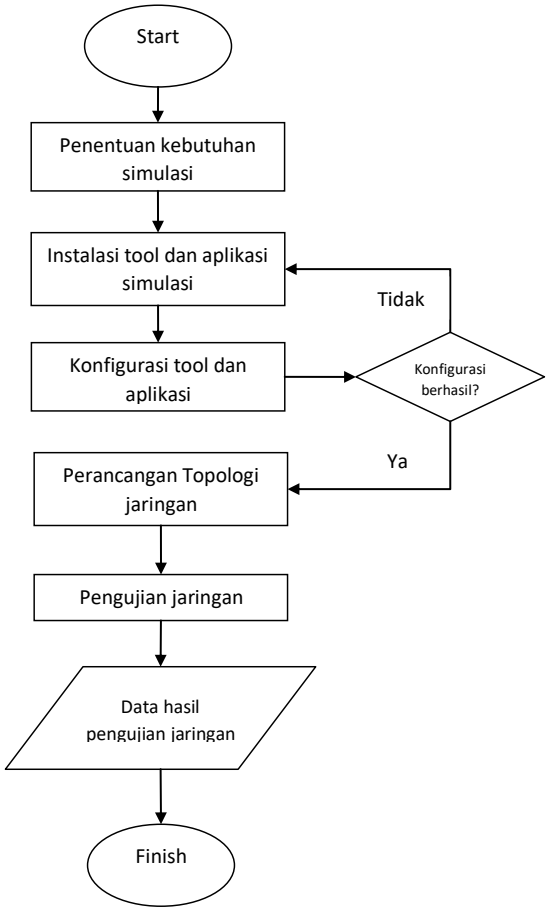
2. Metode Penelitian

a. Studi literatur

Pada Tahap ini dilakukan studi literatur berupa jurnal, e-book, artikel ilmiah, serta sumber dari situs internet yang berkaitan dengan konsep *Software Defined Network* (SDN), konsep *OpenFlow*, parameter pengujian performasi jaringan, serta konfigurasi *mininet*, *Floodlight* dan *Ryu*.

b. Perancangan dan Analisa Sistem

Pada tahap ini dilakukan perancangan sistem yang berkaitan dengan pengujian performansi controller pada arsitektur jaringan SDN. Adapun langkah-langkah yang dilakukan dalam penelitian ini dengan refrensi dari literatur/jurnal dapat dilihat dalam *flowchart* di bawah ini:



Gambar 4 Perancangan Alur Sistem

Penjelasan langkah-langkah dari alur **Gambar 4** di atas sebagai berikut:

a. Penentuan Sistem Simulasi

Berdasarkan dari konsep SDN yang memisahkan antara *control plane* dan *data plane*, maka host yang digunakan dalam instalasi *controller* SDN (*Floodlight* dan *Ryu*) diletakkan secara terpisah dengan *host* yang digunakan dalam instalasi emulator mininet. *Floodlight* dan *Ryu* digunakan dalam menjalankan fungsi sebagai *Control Plane*. Sedangkan mininet digunakan dalam menajalankan proses *Data Plane*.

b. Instalasi Tools dan Aplikasi Simulasi

Melakukan instalasi OpenJDK 8, kemudian instalasi *Controller Floodlight* dan *Ryu*. Floodlight menggunkan versi 1.2 sedangkan *Ryu* menggunakan versi 4.11. Kemudian melakukan instalasi mininet versi 2.2.1. Instalasi mininet dilakukan pada *Virtual Machine* (VM) yang berbeda dengan VM yang digunakan oleh *Floodlight* dan *Ryu*.

c. Konfigurasi Tools dan Aplikasi Simulasi

Mengkonfigurasi server mininet agar dapar terhubung dengan *server* pada controller SDN. Konfigurasi dapat dinyatakan berhasil apabila saat server mininet menjalankan suatu topologi jaringan, pada *server* controller akan terdeteksi identitas perangkat jaringan seperti MAC address pada *switch* yang telah dibuat oleh *mininet*. Apabila controller tidak dapat mendeteksi identitas perangkat jaringan maka proses instalasi akan diulang.

d. Membuat Topologi Pada Mininet

Menbuat topologi pada sisi *host mininet* dengan menghubungkan ke ip controller. Setelah topologi terbentuk maka dilakukan *ping testing* terhadap semua *host*. Jika *ping testing* berhasil maka topologi siap untuk digunakan.

e. Perancangan Topologi Jaringan

Setelah instalasi dan konfigurasi selesai dilakukan, selanjutnya merancang topologi yang akan digunakan dalam pengujian. Topologi yang akan digunakan yaitu topologi *linear* dan topologi mesh. Topologi *mesh* dibuat dengan Bahasa pemrograman python yang akan di implementasikan menggunakan mininet. Sedangkan topologi linier di implementasikan dengan menggunakan fitur tipe topologi yang ada di dalam mininet. Jumlah switch yang akan digunakan mulai dari 4, 8, 12 dan 16 *switch*.

c. Pengujian

Pengujian yang dilakukan dalam jaringan *Software Defined Network* (SDN) ini menggunakan parameter throughput, latency, jitter dan packet loss. Pengambilan Data *Quality of Services* (QoS) dilakukan pada 8 topologi dengan variasi *switch*, yaitu 4, 8, 12 dan 16 *Switch*, dengan *detail host* yang digunakan sebagai *source* dan *destination* untuk masing-masing topologi adalah sebagai berikut:

Tabel 1 Detail Host Source *dan* Destination *Pada Topologi* Linear *dan* Mesh

Topologi Linear / Mesh	Awal		Tujuan	
	Host	IP Address	Host	IP Address
4 Switch	Host-1	10.0.0.1	Host-4	10.0.0.4
8 Switch	Host-1	10.0.0.1	Host-8	10.0.0.8
12 Switch	Host-1	10.0.0.1	Host-12	10.0.0.12
16 Switch	Host-1	10.0.0.1	Host-16	10.0.0.16

Masing-masing pengujian akan dilakukan dari node *source* ke node *destination* seperti tampak pada **Tabel 1**. Pengujian ini berlangsung selama 1 menit sebanyak 3 kali, kemudian nantinya akan diambil nilai rata-rata dari pengujian tersebut. Setiap link di dalam masing-masing topologi memiliki bandwidth sebesar 100 Mbps. Kondisi traffic selama pengujian berlangsung akan menggunakan *background traffic* data yang berkisar mulai dari 50 Mbps, 100 Mbps, 150 Mbps , dan 200 Mbps dari *node* awal ke *node* tujuan yang sama.

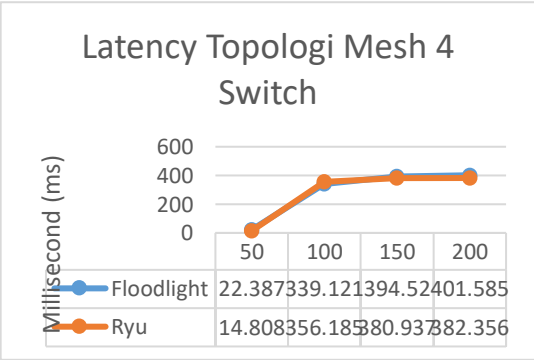
3. Hasil Penelitian dan Pembahasan

a. Pengujian Terhadap Jenis Topologi dan Jumlah *Switch*

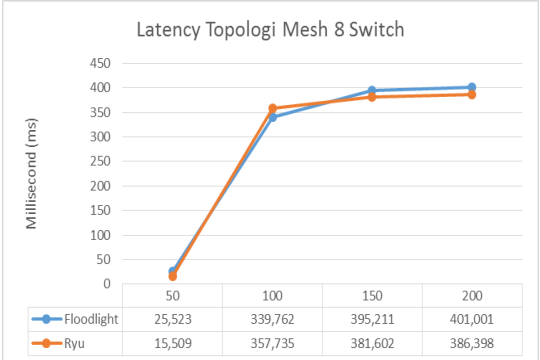
Pada pengujian ini, akan dilakukan pengujian terhadap jenis topologi dengan jumlah *switch* tertentu terhadap performa *controller*. Jumlah *switch* yang digunakan yaitu 4, 8, 12 dan 16 switch dengan masing-masing *switch* memiliki 1 *host*. Pengujian dilakukan dengan mengirimkan *traffic* data dengan *Inter Departure Time* (IDT) konstan sebanyak 1000 pps. Selama 1 menit sebanyak 3 kali. Selama proses berlangsung juga di berikan *background traffic* mulai dari 50 - 200 Mbps.

1) *Latency*

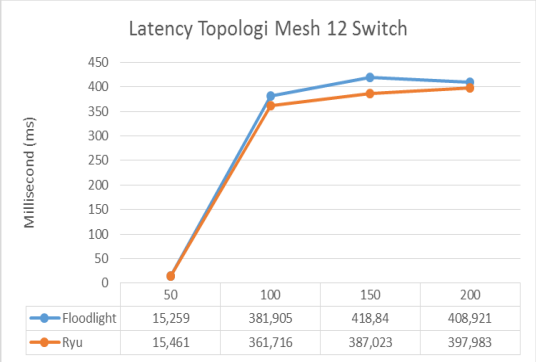
Pada grafik berikut merupakan hasil dari pengukuran *latency* pada topologi *Mesh* dan *Linear*.



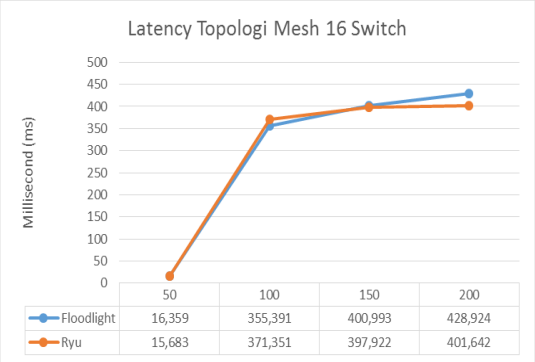
Gambar 5 **Latency *Topologi* Mesh *4* Switch**



Gambar 6 **Latency *Topologi* Mesh *8* Switch**

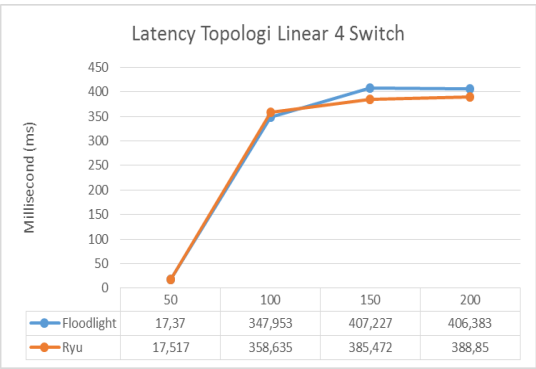


Gambar 7 **Latency *Topologi* Mesh *12* Switch**

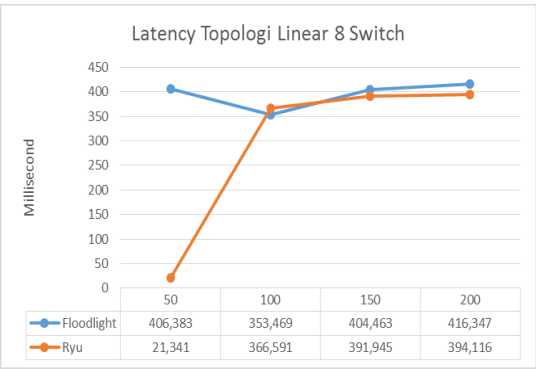


Gambar 8 **Latency *Topologi* Mesh *16* Switch**

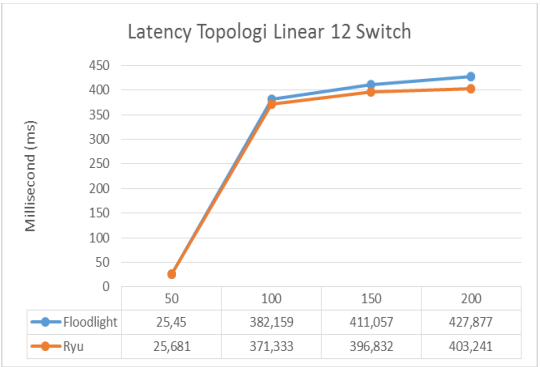
Pada **Gambar 5, 6, 7, 8** hasil pengujian terlihat bahwa nilai *Latency* pada topologi *mesh* yang didapatkan cenderung meningkat ketika dialiri *background traffic* sebesar 100 Mbps baik pada *floodlight* maupun pada *ryu*. Dalam hal ini *controller floodlight* cenderung memiliki nilai *latency* yang lebih besar dibandingkan dengan *ryu* ketika pada jumlah *switch* 4, 8, 12 dan 16 *switch.*



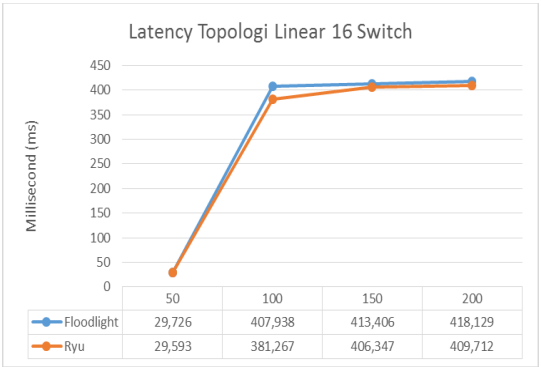
Gambar 9 **Latency *Topologi* Linear *4* Switch**



Gambar 10 **Latency *Topologi* Linear *8* Switch**



Gambar 11 **Latency *Topologi* Linear *4* Switch**

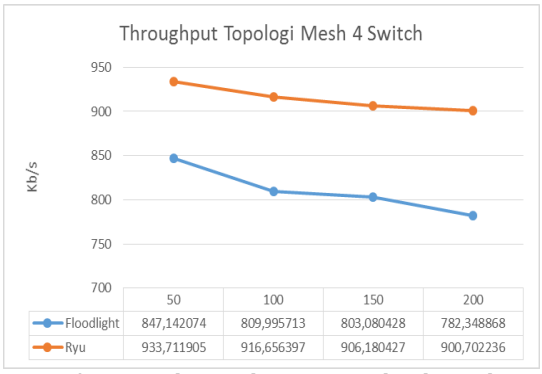


Gambar 12 **Latency *Topologi* Linear *8* Switch**

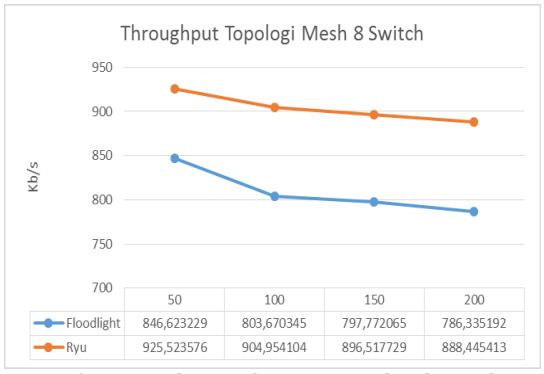
Pada **Gambar 9, 10, 11, 12** hasil pengujian terlihat bahwa nilai *Latency* pada topologi *linear* yang didapatkan sama halnya dengan yang terjadi pada topologi *mesh* yang cenderung meningkat ketika *background traffic* sebesar 100 Mbps Dalam hal ini *controller floodlight* juga cenderung memiliki nilai *latency* yang lebih besar dibandingkan dengan *ryu* ketika pada jumlah *switch* 4, 8, 12 namun ketika pada 16 *switch* nilai *latency* hampir mendekati sama walaupun nilai *latency* pada *floodlight* cenderung lebih besar.

2) *Throughput*

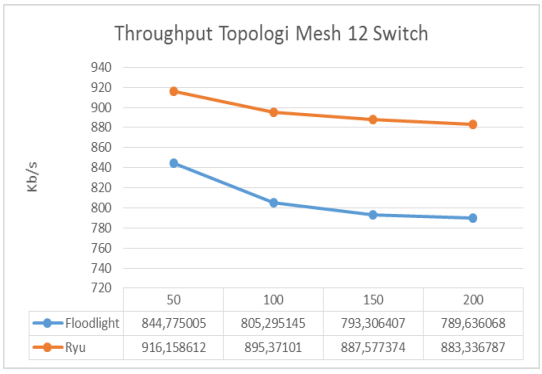
Pada grafik berikut merupakan hasil dari pengukuran *throughput* pada topologi *Mesh* dan *Linear*.



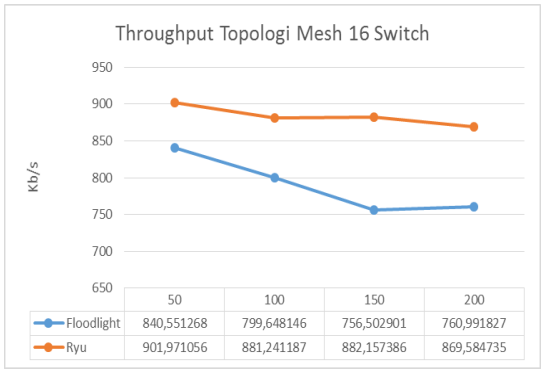
Gambar 13 **Throughput *Topologi* Mesh *4* Switch**



Gambar 14 **Throughput *Topologi* Mesh *8* Switch**

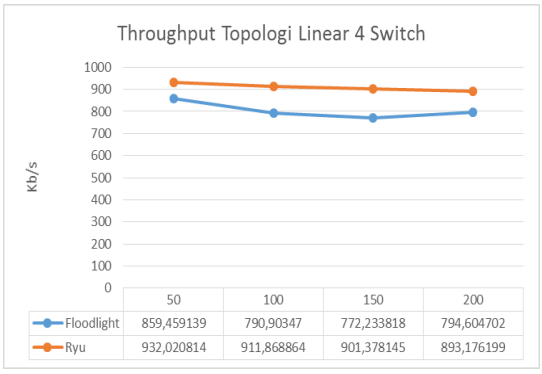


Gambar 15 **Throughput *Topologi* Mesh *12* Switch**

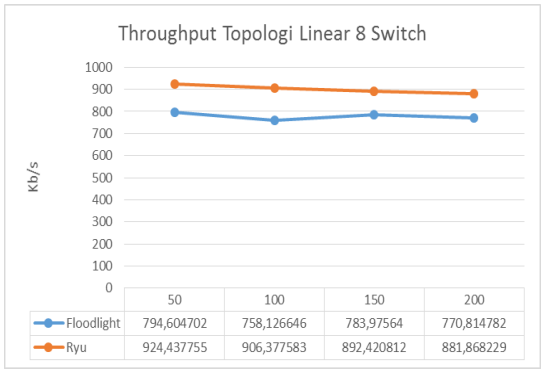


Gambar 16 **Throughput *Topologi* Mesh *16* Switch**

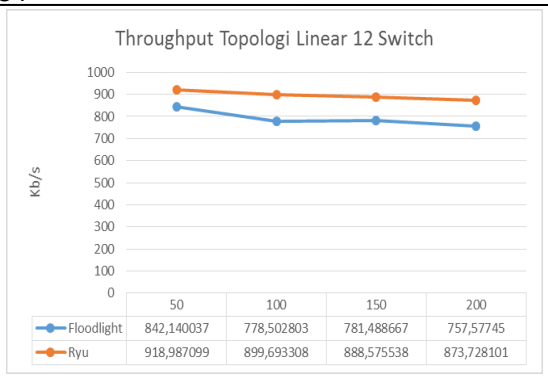
Pada **Gambar 13, 14, 15, 16** hasil pengujian terlihat bahwa nilai *Throughput* pada topologi *mesh* yang didapatkan oleh *controller ryu* lebih besar dibandingkan *controller floodlight.* Sedangkan pada *controller floodlight* cenderung memiliki nilai *throughput* yang lebih rendah, serta cenderung menurun ketika dialiri *background traffic* 150 Mbps pada jumlah 16 *switch*.



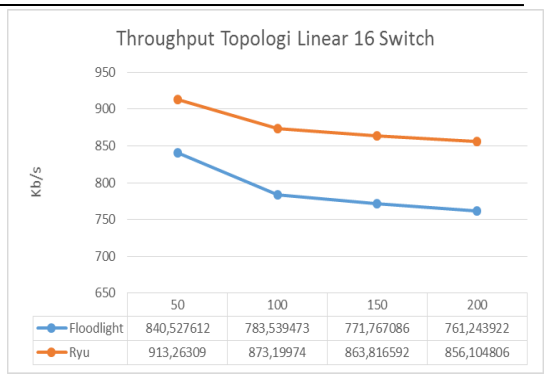
Gambar 17 **Throughput *Topologi* Linear *4* Switch**



Gambar 18 **Throughput *Topologi* Linear *8* Switch**



Gambar 19 **Throughput *Topologi* Linear *12* Switch**

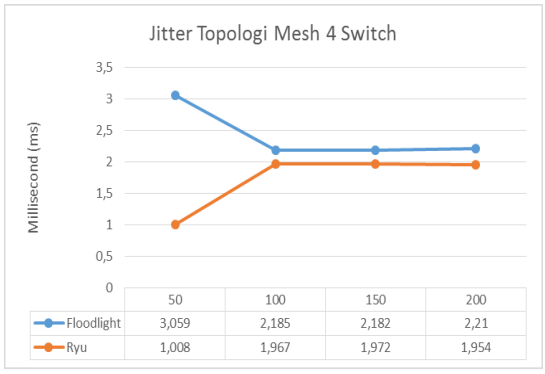


Gambar 20 **Throughput *Topologi* Linear *16* Switch**

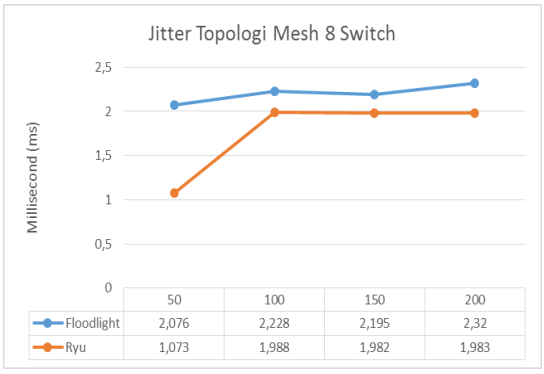
Sedangkan pada **Gambar 17, 18, 19, 20** hasil pengujian terlihat bahwa nilai *Throughput* pada topologi *linear* oleh *controller ryu* juga lebih besar dibandingkan *controller floodlight.* Namun pada saat jumlah 16 *switch,* baik *controller ryu* maupun *floodlight* mengalami penurunan nilai *throughput* ketika dimulai dengan *background traffic* 100 Mbps Meskipun mengalami penurunan, nilai *throughput* yang dihasilkan oleh *controller ryu* masih lebih besar dibandingkan *floodlight.*

3) *Jitter*

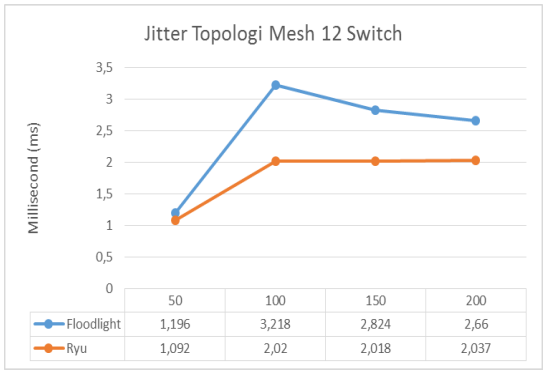
Pada grafik berikut merupakan hasil dari pengukuran *jitter* pada topologi *Mesh* dan *Linear*.



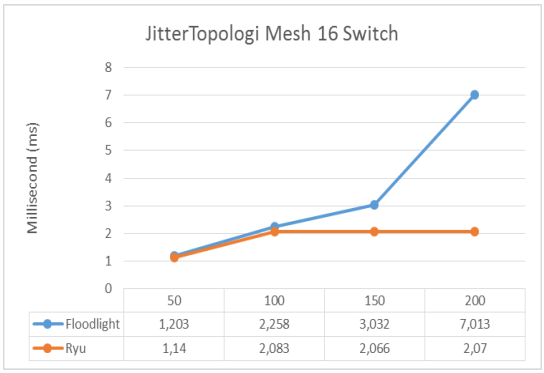
Gambar 21 **Jitter *Topologi* Mesh *4* Switch**



Gambar 22 **Jitter *Topologi* Mesh *8* Switch**

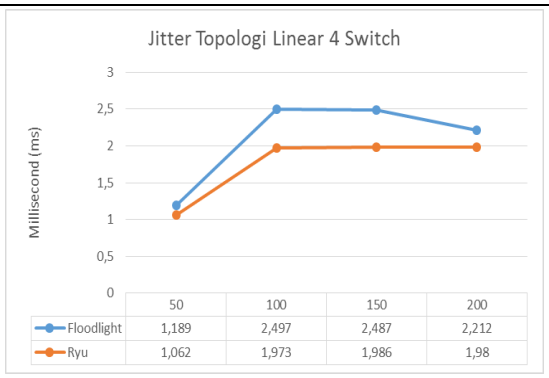


Gambar 23 **Jitter *Topologi* Mesh *12* Switch**

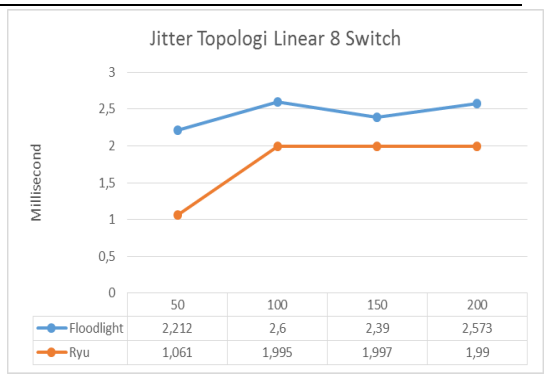


Gambar 24 **Jitter *Topologi* Mesh *16* Switch**

Pada **Gambar 21, 22, 23, 24** hasil pengujian terlihat bahwa nilai *Jitter* pada topologi *mesh* mulai mengalami *kenaikan* ketika dialiri *background traffic* 100 Mbps. Terlihat bahwa ketika beban 100 Mbps, nilai *jitter* mengalami sedikit peningkatan namun cenderung stabil ketika di atas 100 Mbps. *Controller floodlight* memiliki nilai *jitter* yang lebih besar daripada *jitter* pada *ryu.*



Gambar 25 **Jitter *Topologi* Linear *4* Switch**



Gambar 26 **Jitter *Topologi* Linear *8* Switch**

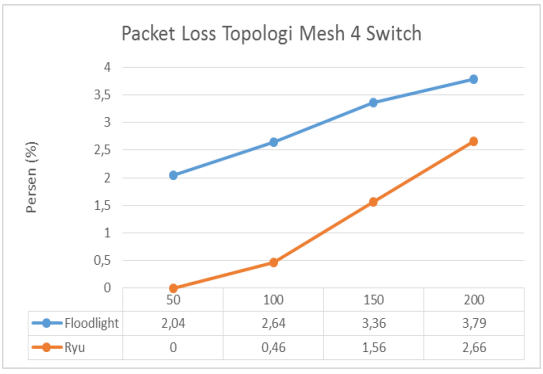
Gambar 27 **Jitter *Topologi* Linear *12* Switch**

Gambar 28 **Jitter *Topologi* Linear *16* Switch**

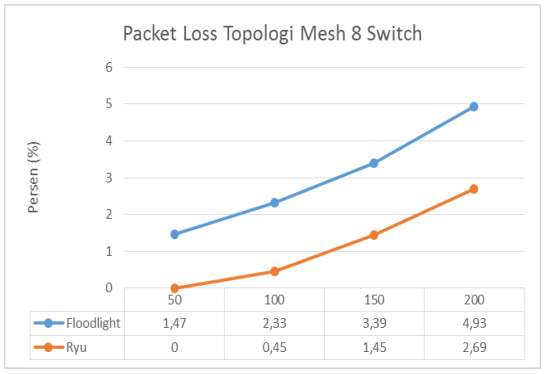
Sedangkan pada **Gambar 25, 26, 27, 28** hasil pengujian terlihat bahwa nilai *Jitter* pada topologi *linear* pada kedua *controller* sama seperti pada topologi *mesh* mengalami kenaikan ketika diberikan beban 100 Mbps dan cenderung stabil ketika di atas 100 Mbps. *Controller floodlight* juga memiliki nilai *jitter* yang lebih besar daripada *jitter* pada *ryu.*

4) *Packet Loss*

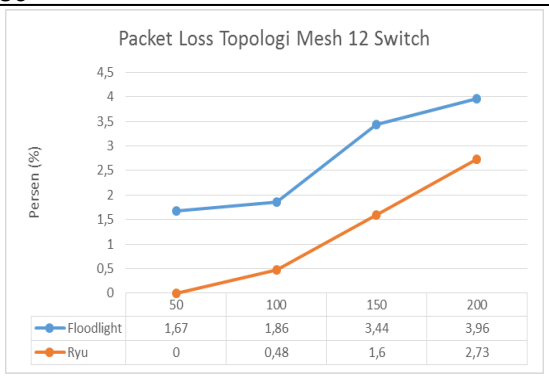
Pada grafik berikut merupakan hasil dari pengukuran *packet loss* pada topologi *Mesh* dan *Linear*.



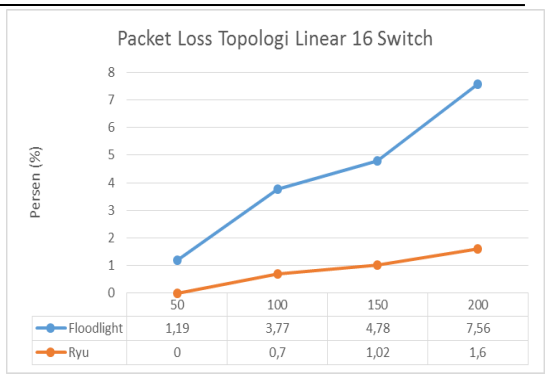
Gambar 29 **Packet Loss *Topologi* Mesh *4* Switch**



Gambar 30 **Packet Loss *Topologi* Mesh *8* Switch**

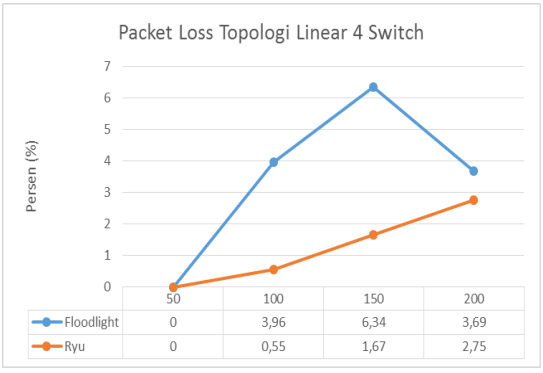


Gambar 31 **Packet Loss *Topologi* Mesh *12* Switch**

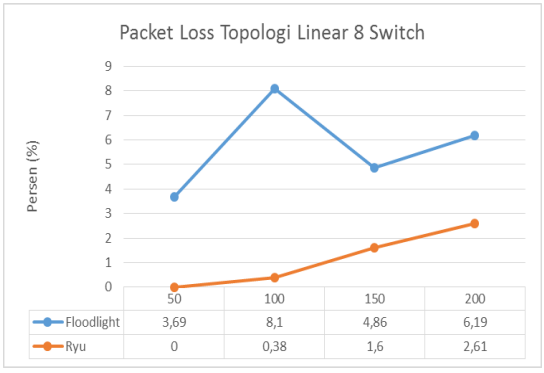


Gambar 32 **Packet Loss *Topologi* Mesh *16* Switch**

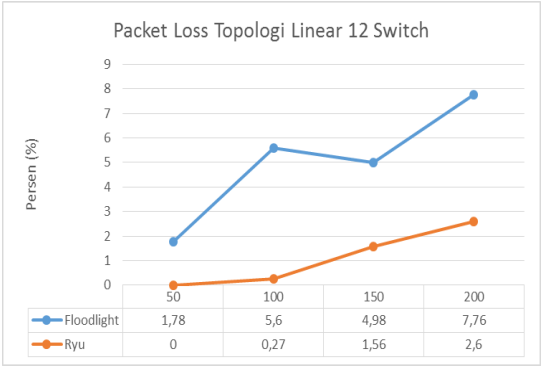
Pada **Gambar 29, 30, 31, 32** hasil pengujian terlihat bahwa nilai *Packet Loss* pada topologi *mesh* baik pada *controller floodlight* maupun *ryu* mengalami peningkatan pada tiap-tiap jumlah *switch.* Dalam hal ini *packet los* yang didapat oleh *floodlight* lebih besar daripada *ryu*.



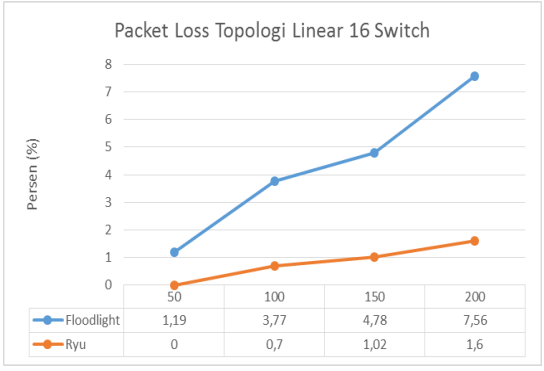
Gambar 33 **Packet Loss *Topologi* Linear *4* Switch**



Gambar 34 **Packet Loss *Topologi* Linear *8* Switch**



Gambar 35 **Packet Loss *Topologi* Linear *12* Switch**



Gambar 36 **Packet Loss *Topologi* Linear *16* Switch**

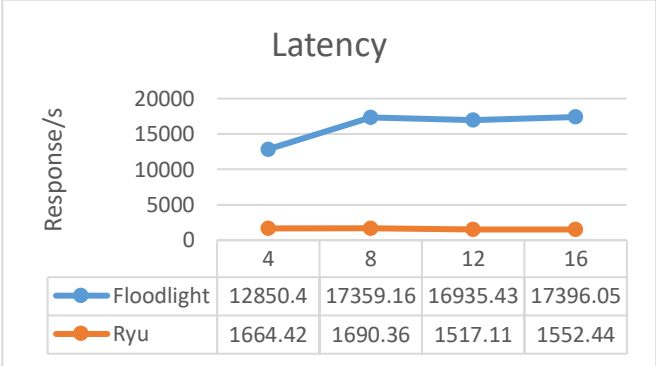
Pada **Gambar 33, 34, 35, 36** hasil pengujian terlihat bahwa nilai *Packet Loss* pada topologi *linear* sama *seperti* yang terjadi pada topologi *mesh,* dimana nilai *packet loss* mengalami kenaikan pada setiap beban *traffic* yang diberikan. Pada topologi *linear* nilai *packet loss* pada *controller floodlight* lebih besar sedangkan pada *controller ryu* memiliki nilai *packet loss* yang cenderung lebih kecil.

b. Pengujian Hanya Terhadap Jumlah *Switch*

Pada pengujian ini, akan dilakukan pengujian hanya terhadap jumlah *switch* tertentu terhadap performa *controller*. Jumlah *switch* yang digunakan yaitu 4, 8, 12 dan 16 *switch* dengan masing-masing *switch* memiliki 1 *host.* Pengujian ini untuk mengukur tingkat aliran (*flow*) data yang dapat ditangani serta tingkat *response* yang dapat diberikan oleh *controller* pada setiap detiknya dengan parameter *Latency* dan *Throughput*.

1) *Latency*

Pengujian *latency* pada *controller floodlight* dan *ryu* dengan jumlah *switch* 4, 8, 12 dan 16 serta masing-masing switch memiliki 1 *host*.

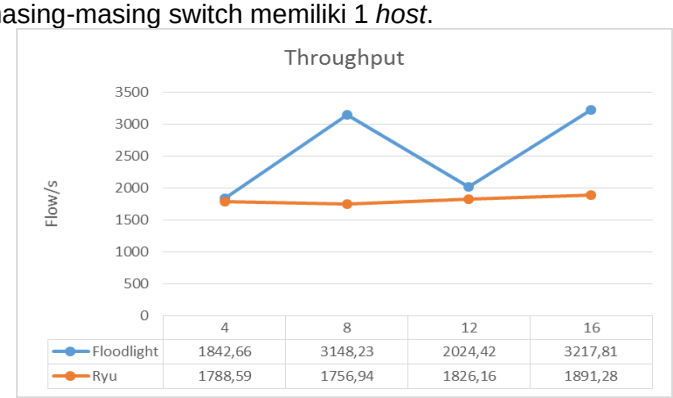


Gambar 37 Pengujian Latency *pada* Controller Floodlight *dan* Ryu

Pada **Gambar 37** terlihat bahwa nilai *latency* pada *controller floodlight* lebih tinggi jika dibandingkan pada *controller ryu* menunjukkan bahwa respons yang dapat diberikan oleh *floodlight* lebih banyak tiap detiknya. Sedangkan *ryu* meskipun memiliki nilai *latency* yang jauh lebih kecil, namun *ryu* mampu memberikan nilai rata-rata yang lebih stabil pada semua *switch* yang diujikan.

2) *Throughput*

Pengujian *latency* pada *controller floodlight* dan *ryu* dengan jumlah *switch* 4, 8, 12 dan 16 serta masing-masing switch memiliki 1 *host*.



Gambar 38 Pengujian Throughput *pada* Controller Floodlight *dan* Ryu

Pada **Gambar 38** terlihat bahwa nilai *throughput* pada *controller floodlight* lebih tinggi dibandingkan dengan *ryu.* Hal ini menunjukkan bahwa jumlah aliran data yang dapat ditangani oleh *floodlight* lebih besar setiap detiknya. Sedangkan *ryu* walaupun memiliki nilai *throughput* yang lebih rendah namun *ryu* dapat tetap menstabilkan nilai *throughput* pada semua *switch* yang telah diujikan.

4. Kesimpulan

Pada pengujian yang telah dilakukan dapat disimpulkan bahwa *controller ryu* memiliki performa yang lebih baik daripada *floodlight* terlihat dari nilai *QoS* ketika diujikan pada topologi *mesh* dan *linear.* Terlihat pada nilai *latency* meskipun terjadi perubahan yang signifikan ketika dialiri *traffic* 100 Mbps, *floodlight* memiliki nilai yang lebih tinggi. Untuk hasil *throughput*, *ryu* memiliki nilai yang lebih tinggi dari *floodlight* dengan kisaran nilai 856-933 Kbps. . Hasil nilai *jitter* yang dihasilkan pada *floodlight* lebih tinggi namun nilainya cenderung tidak stabil Sedangkan nilai *jitter* pada *ryu* cenderung stabil hanya meningkat ketika *traffic* 100. Hasil *packet loss* pada *floodlight* lebih tinggi jika dibandingkan dengan ryu. Controller ryu hanya memiliki rata-rata nilai *packet loss* sebesar 0,5% untuk semua jenis topologi dan *background traffic*. Pada pengujian hanya terhadap jumlah *switch*, *floodlight* memiliki performa yang lebih baik. *Floodlight* menjamin terhadap tingkat respons yang lebih banyak serta pengelolaan data yang besar. Sedangkan *ryu* menjamin dalam tingkat respons serta pengelolaan data yang lebih stabil di dalam arsitektur jaringan SDN.

Refrensi

[1] S. Ramadona, B. A. Hidayatulloh, D. F. Siswanto, and N. Syambas, “The simulation of SDN network using POX controller: Case in Politeknik Caltex Riau,” *Proceeding 2015 9th Int. Conf. Telecommun. Syst. Serv. Appl. TSSA 2015*, 2016.

[2] Sucres, “Software-Defined Networking : Past, Present, and Future of Programmable Networks,” *2016-05-06*, 2016. [Online]. Available: https://www.sucrez.com/software-defined-networking/.

[3] G. Romero de Tejada Muntaner and G. R. D. T. Muntaner, “Evaluation of OpenFlow Controllers,” p. 90, 2012.

[4] Project Floodlight, “Floodlight OpenFlow Controller -Project Floodlight,” 2013. [Online]. Available: http://www.projectfloodlight.org/floodlight/.

[5] Ryu SDN Framework Community, “Ryu SDN Framework.” [Online]. Available: https://osrg.github.io/ryu/.

[6] Mininet, “Mininet Overview - Mininet,” 2017. [Online]. Available: http://mininet.org/overview/.

[7] A. METZLER and A. Metzler, “Ten Things to Look for in an SDN Controller,” p. 11, 2013.

[8] R. Kartadie and B. Satya, “Uji Performa Kontroler Floodlight Dan Opendaylight Sebagai Komponen Utama Arsitektur Software-Defined Network,” *Semnasteknomedia Online*, no. June, 2015.

[9] S. M. Anggara, “Pengujian Performa Kontroler Software-defined Network (SDN): POX dan Floodlight 2012/2 013,” 2015.

[10] Y. Zhao, L. Iannone, and M. Riguidel, “On the performance of SDN controllers: A reality check,” *2015 IEEE Conf. Netw. Funct. Virtualization Softw. Defin. Network, NFV-SDN 2015*, pp. 79–85, 2016.