Analisis Perbandingan Performansi *Controller* Ryu & POX Pada *Software-Defined Network*

1st Adhitya Insan Nurizky
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
adhityain@student.telkomuniversity.ac.id

2nd Rendy Munadi
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
rendymunadi@telkomuniversity.ac.id

3rd Sofia Naning Hertiana
Fakultas Teknik Elektro
Universitas Telkom
Bandung, Indonesia
sofiananing@telkomuniversity.ac.id

***Abstrak*— Internet menjadi salah satu kebutuhan penting di kehidupan masyarakat Indonesia. Salah satu konsep jaringan yang sedang berkembang adalah *Software-Defined Network*. Pada SDN, terjadi pemisahan antara *control plane* dan *data plane*, *control plane* dipindahkan ke sebuah kontroler jaringan, dan *data plane* tetap berada di *switch*, sehingga pemusatan jaringan dapat terjadi. Terdapat beberapa kontroler yang digunakan pada SDN, salah duanya adalah POX dan Ryu. Penelitian ini bertujuan untuk mengetahui kontroler manakah di antara POX dan Ryu, yang menghasilkan *Quality of Service* yang maksimal, melalui uji coba yang dilakukan pada Mininet. Percobaan dilakukan pada Mininet dengan membandingkan performansi jaringan saat menggunakan kontroler POX dan Ryu. Terdapat tiga skenario percobaan dengan menggunakan 4, 5 dan 6 perangkat *switch* yang masing-masing terhubung dengan 2 host, topologi tersebut menggunakan *bandwidth traffic* sebesar 100 Mbps, dan *background traffic* sebesar 100 dan 200 Mbps. Lalu data penelitian didapatkan menggunakan aplikasi D-ITG untuk parameter-parameter yang diinginkan. Hasil yang didapatkan setelah pengujian adalah, Ryu mendapatkan nilai *delay* 67% lebih rendah dan nilai *jitter* 2,94% lebih rendah dibandingkan POX. Sedangkan POX mendapatkan nilai *throughput* 3,57% lebih tinggi dibandingkan Ryu. Pada *packet loss*, Ryu tidak mengalami *packet loss* sama sekali, sedangkan POX masih mengalami *packet loss* pada skenario 1 dan 2.**

Kata kunci— *Software-Defined Network*, QoS, Mininet, D-ITG.

I. PENDAHULUAN

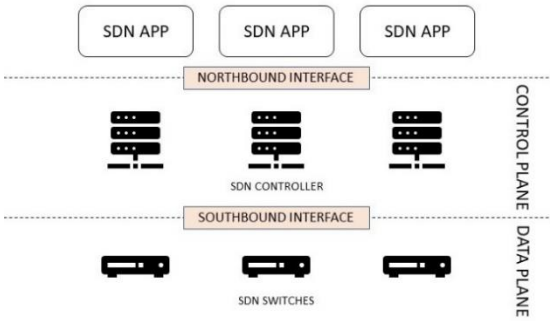
Internet saat ini menjadi salah satu kebutuhan yang penting dalam kehidupan manusia, termasuk di negara Indonesia. Hampir setiap hari masyarakat Indonesia menggunakan internet, sehingga infrastruktur jaringan yang digunakan seiring dengan zaman pun terus berkembang. Salah satu konsep jaringan yang saat ini sedang berkembang adalah *Software-Defined Network* (SDN). Konsep ini memiliki perbedaan dengan konsep jaringan tradisional, yang mana *control plane* dan *data plane* berada pada satu perangkat *switch*. Pada SDN, terjadi pemisahan antara *control plane* dan *data plane*, di mana, *control plane* dipindahkan ke sebuah kontroler jaringan, dan *data plane* tetap berada di *switch*, sehingga pemusatan jaringan dapat terjadi.

Terdapat beberapa kontroler yang dapat digunakan pada SDN, salah duanya adalah POX dan Ryu. Dikarenakan ada banyaknya kontroler yang dapat digunakan, maka untuk memastikan bahwa hasil yang dicapai bisa mencapai maksimal, perlu diadakan perbandingan pada kontroler-kontroler tersebut. Terdapat beberapa penelitian yang melibatkan analisis performansi SDN, dengan berbagai controller, serta berbagai variable Quality of Service. Oleh karena itu, Tugas Akhir ini bertujuan untuk mengetahui kontroler manakah di antara POX dan Ryu, yang dapat menghasilkan *Quality of Service* yang maksimal, melalui uji coba yang akan dilakukan pada Mininet.

II. KAJIAN TEORI

A. Software-Defined Network

Software-Defined Network merupakan sebuah konsep, pengembangan dari jaringan konvensional, yang mana pada jaringan tersebut, *control plane* dan *data plane* ditempatkan pada sebuah perangkat *switch*. Hal ini mengakibatkan beratnya kerja *switch* sehingga performansi dari jaringan tersebut tidak maksimal. Sehingga SDN menawarkan sebuah konsep untuk memisahkan *control plane* dan *data plane*, sehingga dapat tercipta jaringan yang terpusat [4]. *Control plane* yang tadinya berada pada perangkat *switch*, dikeluarkan dan ditempatkan pada sebuah kontroler, yang mana berperan penting dalam arsitektur SDN. Kontroler berperan untuk memberikan sebuah layanan yang dapat merealisasikan *control plane* yang terdistribusi, dan juga memberikan kemampuan untuk mengatur jaringan secara terpusat [4]. Kemudian, *switch* yang sudah tidak memiliki *control plane*, hanya



GAMBAR 1
Arsitektur Software-Defined Network [9]

bertugas sebagai *forwarding packet*, sehingga bisa disebut sebagai perangkat *dummy*.

Berbagai fitur yang ditawarkan oleh Software Defined Network menjadikannya sebagai salah satu unggulan teknologi di masa depan. Beberapa keunggulan SDN yang ditawarkan antara lain [10-11]:

a. Infrastruktur yang siap pakai di masa yang akan datang
b. Mengurangi pengeluaran pada perangkat keras.
c. Mengatur jaringan secara terpusat

B. OpenFlow

OpenFlow adalah protokol yang digunakan oleh SDN, dan didesain untuk dapat memberikan akses kepada aplikasi eksternal kepada perangkat *forwarding* sebuah jaringan, dalam hal ini adalah *switch* [12]. Protokol ini dibentuk oleh sebuah lembaga bernama *Open Networking Foundation* (ONF) pada tahun 2011 [4].

Komponen utama pada OpenFlow, seperti yang dapat dilihat pada Gambar 2.3, antara lain [12]:

a. *OpenFlow channel*
b. *Group table*
c. *Flow table*

Group table dan *Flow table* bertanggungjawab sebagai pencari paket dan juga sebagai *forwarding*. Sedangkan *OpenFlow channel* digunakan untuk berkomunikasi dengan kontroler eksternal. Kontroler lalu menggunakan protokol OpenFlow untuk mengatur beberapa switch [12].

Cara kerja OpenFlow adalah dengan cara mengatur *flow table* pada *switch* untuk dapat bekerja bersama dalam memproses paket yang diterima. Setiap *switch* harus memiliki setidaknya satu *flow table*. *Table* pertama akan dinamakan *Table* 0. Lalu setiap *table* tambahan nantinya akan mempunyai nama sesuai urutannya. Paket yang diterima awalnya akan mempunyai bagian seperti *header* yang akan diperiksa apakah cocok dengan *flow table* 0. Jika sama, maka instruksi yang terdapat pada *flow table* itu akan dijalankan. Hal ini akan terus berjalan seiring paket yang dikirimkan. Proses ini akan berhenti apabila *flow table* tidak lagi memiliki instruksi untuk meneruskan paket yang dikirim ke *flow table* lainnya. Jika hal ini terjadi, maka paket akan diteruskan oleh *switch*.

Tidak semua paket akan cocok dengan *flow table*. Kejadian ini dinamakan dengan *table-miss*. Apabila ini terjadi, maka beberapa tindakan akan otomatis dilakukan, seperti *drop* paket, mencocokkan paket dengan *table* lain, atau mengirimkan paket tersebut ke kontroler [12].

Pada SDN, *switch* terkoneksi dengan kontroler melalui *OpenFlow channel*. *OpenFlow channel* adalah antarmuka yang digunakan kontroler untuk mengkonfigurasi serta mengatur *switch*, menerima *events* serta mengirim paket ke *switch*. Pada *OpenFlow channel*, terdapat tiga pesan yang digunakan oleh kontroler untuk bertukar informasi dengan *switch*, yaitu [12]:

a. *Controller-to-Switch*
   Pesan ini digunakan untuk mengatur atau mengecek secara langsung keadaan *switch* dan pesan ini dikirim oleh kontroler.
b. *Asynchronous*
   Pesan ini digunakan untuk memperbaharui pengaturan pada kontroler sesuai dengan kondisi jaringan yang ada, lalu melakukan perubahan kondisi pada *switch*. Pesan ini dikirimkan oleh *switch*.
c. *Symmetric*
   Pesan ini dapat dikirimkan oleh kedua kontroler ataupun *switch*, dan dapat dikirimkan tanpa diperlukan permohonan sebelumnya

C. Kontroler

Kontroler adalah sebuah entitas pada SDN, yang mana bertugas untuk memberikan layanan untuk dapat merealisasikan pemusatan jaringan. Selain itu, kontroler juga memberikan layanan berupa model data tingkat tinggi yang menghubungkan antara sumber daya dengan berbagai layanan lain yang ditawarkan oleh kontroler. Layanan tersebut ditampilkan melalui sebuah API pada sebuah aplikasi [4].

Terdapat beberapa macam kontroler SDN yang umum digunakan, seperti Ryu, Open DayLight, ONOS, Floodlight dan POX. POX merupakan kontroler SDN yang dibentuk melalui bahasa Python. POX dapat digunakan untuk mengubah perangkat OpenFlow menjadi sebuah *hub*, *switch*, *load balancer*, *firewall*, dan lain lain [13].

D. Mininet

Mininet adalah sebuah alat *emulator* yang memungkinkan pengguna untuk menjalankan sejumlah *host*, *switch* dan kontroler secara *virtual*. Mininet menggunakan konsep virtualisasi berbasis kontainer sehingga membuat sebuah sistem dapat bekerja layaknya seperti jaringan sesungguhnya [14]. Mininet dapat berjalan pada sebuah komputer atau laptop yang menggunakan sistem operasi Linux, dan dapat juga beroperasi dari dalam *virtual machine*, seperti VMware atau Oracle [15].

Adapun beberapa kelebihan dari Mininet, seperti :

a. Dapat membuat sebuah topologi sederhana dengan cepat,
b. Dapat membuat topologi dengan bebas sesuai keinginan,
c. Dapat menjalankan aplikasi didalamnya,
d. Mudah untuk mengatur *forwarding* paket,

e. Dapat dijalankan di berbagai perangkat seperti komputer, laptop, VM, dan lain lain,
f. Hasil pengerjaan dapat disebarluaskan,
g. *Open source*, sehingga mudah untuk dikembangkan.

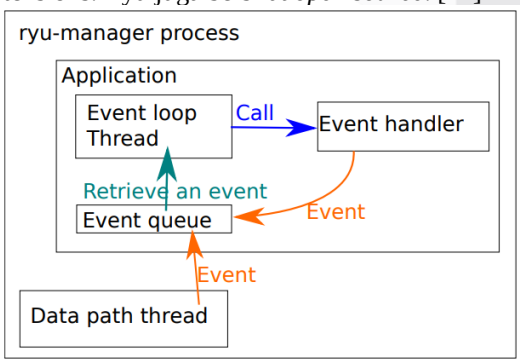
E. POX

POX adalah salah satu jenis kontroler OpenFlow yang tersedia untuk platform Software Defined Network. Berjalan dalam bahasa Python, POX awalnya dibuat hanya sebagai kontroler OpenFlow, namun seiring berjalannya waktu, pengembang dapat membuat POX menjadi salah satu pilihan OpenFlow switch. Untuk dapat berjalan dengan baik, POX membutuhkan setidaknya Python 2.7, dan dapat beroperasi di berbagai macam sistem operasi seperti Linux, Mac OS, dan Windows. Saat ini POX dapat berkomunikasi dengan switch OpenFlow 1.0 dan mendapatkan dukungan layanan pada Open vSwitch/Nicira [16].

POX memiliki beberapa aspek dalam mekanisme kerjanya. Yang pertama adalah bagaimana cara POX bekerja dengan alamat jaringan. IPv4, IPv6 dan alamat Ethernet direpresentasikan dengan kelas-kelas seperti IPAddr, IPAddr6 dan EthAddr dari pox.lib.addresses. Kemudian ada *Event Handling*, yang menggunakan model *publish/subscribe*. Beberapa objek dapat mem-*publish event*, kemudian objek lain dapat melakukan *subscribe* pada objek tersebut secara spesifik. Yang ketiga, adalah bagaimana cara POX bekerja dengan paket. Aplikasi pada POX berinteraksi dengan paket-paket data. Untuk memfasilitasi hal tersebut, POX memiliki *library* untuk menguraikan dan membangun paket. *Library* ini mendukung berbagai macam tipe paket.

F. Ryu

Ryu adalah salah satu jenis kontroler yang digunakan pada *Software-Defined Network*. Sama dengan POX, Ryu juga berjalan dalam bahasa Python. Ryu mendukung beberapa macam protokol untuk mengatur perangkat jaringan, seperti OpenFlow, Netconf, OF-config dan lainnya. Untuk OpenFlow, Ryu mendukung versi 1.0, 1.2, 1.3, 1.4, 1.5 dan Nicira Extensions. Ryu juga bersifat *open source*. [17]



GAMBAR 2
Arsitektur Ryu

Arsitektur pada Ryu terbagi menjadi 5 bagian, yang pertama adalah *Applications*. Pada bagian ini, logika dari pengguna Ryu disebut sebagai sebuah *Application*, yang merupakan sebuah class dari ryu.base.app_manager.RyuApp. Bagian kedua adalah *Event*, yang merupakan sebuah class dari ryu.controller.event.EventBase. Komunikasi yang berlangsung antara *Applications* dilakukan dengan mengirimkan dan menerima *Events*. Kemudian, bagian ketiga adalah Event Queue, yang mana setiap aplikasi memiliki 1 antrian setiap menerima *Events*. Bagian ke empat adalah *Threads*. Ryu berjalan secara *multi-thread* menggunakan eventlets. Dikarenakan *threads* bersifat tidak mendahului, sehingga perlu berhati-hati dalam menjalankan proses yang memakan waktu. Yang kelima, adalah *Event Handlers*, yang dapat didefinisikan dengan menambahkan ryu.controller.handler.set_ev_cls pada *class* aplikasi. Ketika suatu event spesifik terjadi, *event handler* akan dipanggil dari *event loop* aplikasi.

G. User Datagram Protocol (UDP)

User Datagram Protocol, atau yang biasa dikenal dengan UDP, adalah sebuah protokol pada transport layer yang bersifat *connectionless*, yang artinya adalah komunikasi antara dua titik pada suatu jaringan dapat berjalan tanpa perlu adanya komunikasi dua arah antara titik tersebut.

Paket UDP, yang bisa disebut dengan User Datagram, memiliki header dengan ukuran 8 bytes, yang terbagi menjadi beberapa bagian:

a. Nomor port sumber

Merupakan nomor port yang digunakan oleh proses yang berjalan di host sumber. Memiliki panjang 16 bit, yang berarti nomor port bervariasi dari 0 sampai 65.535.

b. Nomor port tujuan

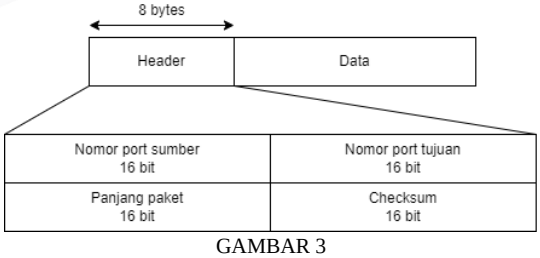
Sama seperti nomor port sumber, namun nomor ini merupakan nomor port yang digunakan pada host tujuan. Memiliki panjang yang sama yaitu 16 bit.

c. Panjang paket

Panjang paket pada UDP adalah 16 bit, yang berarti memiliki total panjang yang bervariasi dari 0 sampai 65.535 bytes. Namun, total panjang paket UDP harus jauh lebih kecil karena UDP tersimpan pada datagram IP dengan total panjang 65.535 bytes.

d. Checksum

Checksum digunakan untuk mendeteksi error pada keseluruhan user datagram (header dan data).



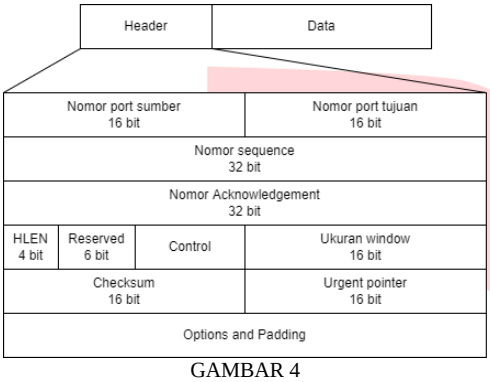
GAMBAR 3
Format UDP

Ada berbagai penggunaan UDP yang umum diketahui, diantaranya, UDP cocok digunakan untuk proses yang membutuhkan komunikasi request-response yang simple. Selain itu, UDP juga cocok

digunakan untuk protokol transport untuk multicasting. Dan UDP juga dapat digunakan untuk manajemen proses seperti SNMP.

H. Transmission Control Protocol (TCP)

Transmission Control Protocol, atau yang biasa disebut dengan TCP, merupakan protokol process-to-process. Seperti UDP, TCP juga menggunakan nomor port. Namun, perbedaan dengan UDP adalah, TCP merupakan protokol connection-oriented, yang artinya, TCP membuat sebuah koneksi virtual antara dua host dengan protokol TCP untuk mengirim data.



GAMBAR 4
Format TCP

Pada TCP, header memiliki ukuran 20-60 bytes, yang terbagi menjadi beberapa bagian:

a. Nomor port sumber
Merupakan nomor port yang digunakan oleh proses yang berjalan di host sumber. Memiliki panjang 16 bit, yang berarti nomor port bervariasi dari 0 sampai 65.535.

b. Nomor port tujuan
Sama seperti nomor port sumber, namun nomor ini merupakan nomor port yang digunakan pada host tujuan. Memiliki panjang yang sama yaitu 16 bit.

c. Nomor sequence
Nomor sequence menjelaskan nomor yang sudah ditetapkan pada byte pertama pada sebuah data yang tergabung pada suatu segment.

d. Nomor acknowledgement
Nomor ini menjelaskan byte yang akan diterima oleh penerima segment dari host lain. Apabila penerima akan menerima byte dengan nomor x, maka nomor acknowledgementnya adalah x+1.

e. Panjang header (HLEN)
Mengindikasikan nomor panjang header. Panjang header bisa bervariasi antara 20-60 bytes, sehingga nilai dari HLEN akan berada di antara 5 (5 x 4 = 20) dan 15 (15 x 4 = 60).

f. Reserved
Bagian ini memiliki 6 bit yang disimpan apabila di butuhkan.

g. Control
Bagian ini memiliki 6 bit control yang berbeda:

GAMBAR 5
Bagian control TCP

TABEL 1
Deskripsi control TCP

Flag	Deskripsi
URG	Nilai pada urgent pointer valid
ACK	Nilai pada nomor acknowledgement valid
PSH	Push data
RST	Me-reset koneksi
SYN	Melakukan sinkronisasi nomor sequence
FIN	Memutus koneksi

h. Ukuran window
Bagian ini mendefinisikan ukuran window dengan ukuran 16 bit. Yang artinya, ukuran maksimumnya adalah 65.535 bytes. Nilai ini ditentukan oleh penerima paket, yang harus diikuti oleh pengirim.

i. Checksum
Bagian ini memiliki definisi yang sama dengan UDP, namun, pada paket UDP, checksum bersifat opsional. Sedangkan pada TCP, checksum bersifat wajib.

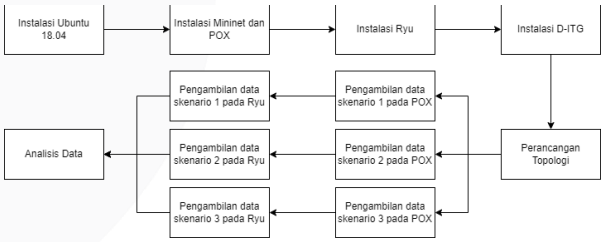
j. Urgent Pointer
Bagian ini hanya valid apabila flag URG pada control telah diset. Digunakan ketika segment memiliki data yang bersifat urgent.

k. Options
Bagian ini berisikan informasi opsional pada TCP, yang dapat memuat informasi mencapai 40 bytes.

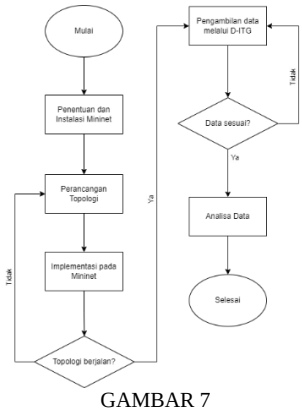
III. METODE

A. Desain Sistem

Berikut adalah diagram blok dan alir yang menggambarkan secara umum mengenai system yang dirancang untuk penelitian ini.



GAMBAR 6
Diagram Blok Sistem



GAMBAR 7

Diagram Alir Sistem

B. Desain Perangkat Keras

Pada penelitian ini, terdapat perangkat keras yang digunakan. Adapun perangkat-perangkat yang digunakan adalah sebagai berikut:

TABEL 2
Perangkat Keras dan Fungsinya

No.	Komponen	Keterangan
1.	CPU	Intel Core i3-7020U @ 2.3 GHz
2.	RAM	12 GB

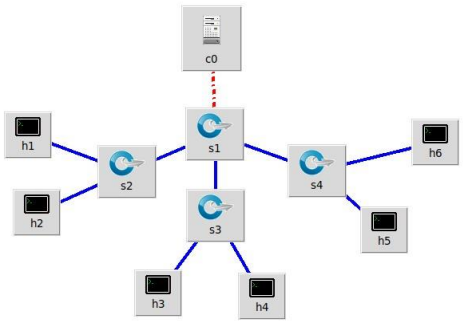
C. Desain Perangkat Lunak

Pada penelitian ini, terdapat beberapa perangkat lunak yang digunakan untuk mendukung kelancaran sistem. Adapun perangkat lunak yang digunakan adalah sebagai berikut.

TABEL 3
Perangkat Lunak dan Fungsinya

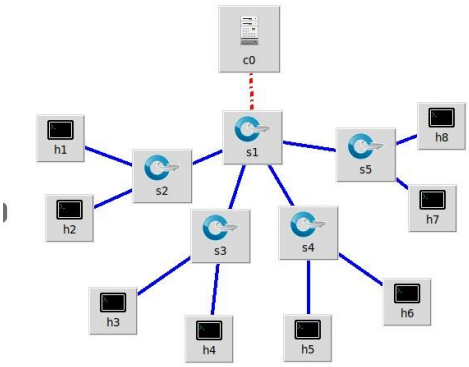
No.	Perangkat Lunak	Fungsi
1.	Kontroler POX dan Ryu	Sebagai *control plane*
2.	Mininet	Sebagai *emulator* jaringan yang akan didesain
3.	D-ITG	Untuk melihat data penelitian
1	VirtualBox 6.1.14	Menjalankan sistem operasi Ubuntu 18.14
5.	Ubuntu 18.04	Sebagai sistem operasi pada penelitian
6.	Iperf	Untuk membangkitkan trafik

D. Perancangan Topologi



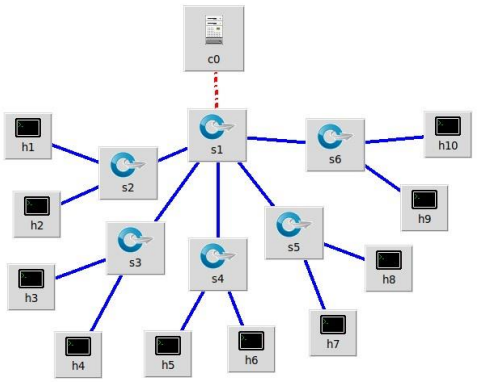
GAMBAR 8
Skenario topologi 1

Gambar 8 di atas memperlihatkan bahwa terdapat 4 *switch*, dengan *switch* 1 berperan sebagai *central switch* dan terhubung dengan 3 *switch* lainnya, serta terhubung dengan kontroler secara langsung. Sedangkan *switch* 2 sampai 4 masing-masing terhubung dengan 2 *host*.



GAMBAR 9
Skenario topologi 2

Gambar 9 di atas memperlihatkan bahwa terdapat 5 *switch*, dengan *switch* 1 berperan sebagai *central switch* dan terhubung dengan 4 *switch* lainnya, serta terhubung dengan kontroler secara langsung. Sedangkan *switch* 2 sampai 5 masing-masing terhubung dengan 2 *host*.



GAMBAR 10
Skenario topologi 3

Gambar 10 di atas memperlihatkan bahwa terdapat 6 *switch*, dengan *switch* 1 berperan sebagai *central switch* dan terhubung dengan 5 *switch* lainnya, serta terhubung dengan kontroler secara langsung. Sedangkan *switch* 2 sampai 6 masing-masing terhubung dengan 2 *host*.

E. Skenario Pengujian

Penelitian akan dimulai dengan membuat topologi yang akan digunakan pada Mininet. Pembuatan topologi ini dilakukan dengan cara membuat 3 file python pada direktori mininet/custom, dengan nama file skenario1.py, skenario2.py dan skenario3.py. Kemudian, percobaan pertama akan dilakukan menggunakan kontroler POX pada ke 3 skenario tersebut. Dilakukan pengumpulan data *Quality of Service* (QOS) seperti *delay*, *jitter*, *throughput* dan *packet loss* pada paket data UDP yang diambil melalui D-ITG. Hal ini dilakukan setelah menaikkan *background traffic* sebesar 100 Mbps dan 200 Mbps

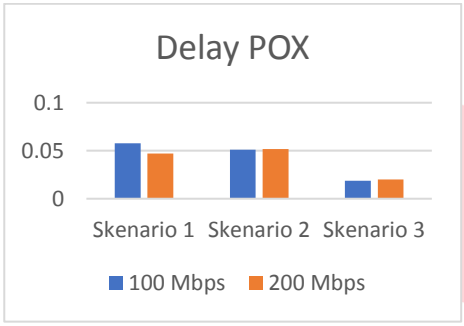
Setelah itu, percobaan kedua akan dilakukan menggunakan kontroler Ryu pada ke 3 skenario tersebut. Data yang dikumpulkan masih sama, yaitu *delay*, *jitter*, *throughput* dan *packet loss* pada paket

data UDP melalui D-ITG. Percobaan dilakukan pada background traffic sebesar 100 dan 200 Mbps.

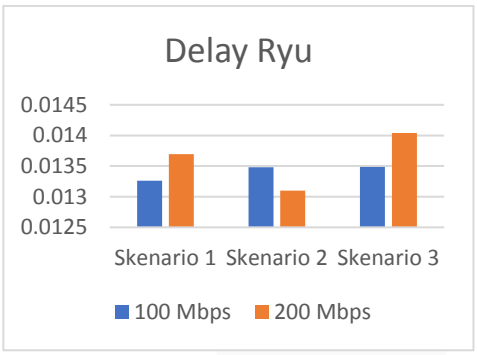
Setelah mendapatkan seluruh data-data yang dibutuhkan, dilakukan perbandingan menggunakan grafik untuk mendapatkan hasil, yaitu kontroler manakah yang berjalan lebih efektif antara POX dan Ryu pada *Software-Defined Network*.

IV. HASIL DAN PEMBAHASAN

A. Delay



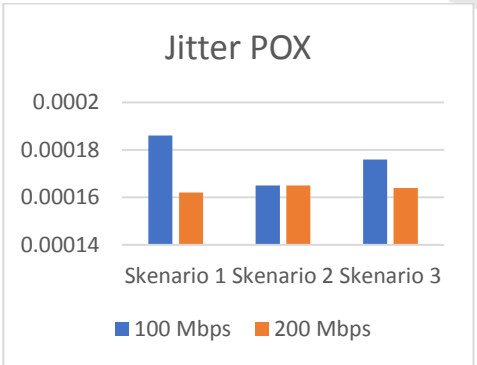
GAMBAR 11
Grafik *delay* pada kontroler POX



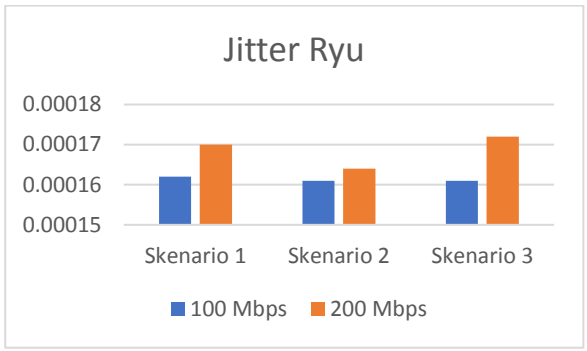
GAMBAR 12
Grafik *delay* pada kontroler Ryu

Seperti yang bisa dilihat pada gambar 11 dan 12, terdapat perbedaan yang signifikan pada *delay* yang dialami oleh kedua kontroler tersebut. Kontroler Ryu mengalami rata-rata *delay* yang lebih singkat sebesar 67% pada ketiga skenario pengetesan yang dilakukan, dibandingkan dengan kontroler POX.

B. Jitter



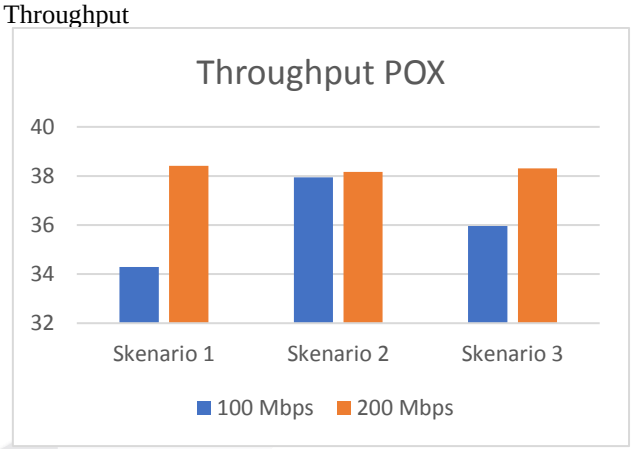
GAMBAR 13
Grafik *jitter* pada kontroler POX



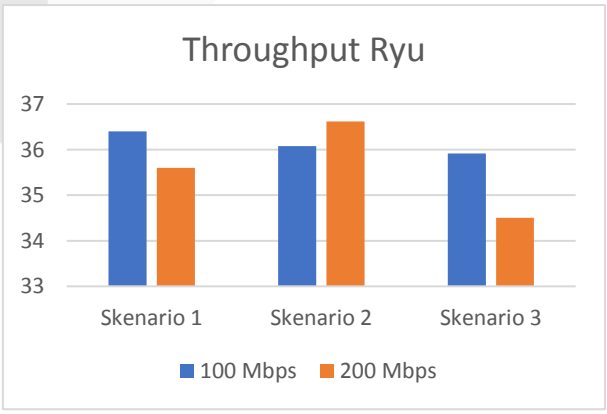
GAMBAR 14
Grafik *jitter* pada kontroler Ryu

Berdasarkan gambar 13 dan 14, dapat dilihat kedua kontroler memiliki angka yang mirip. Angka yang didapatkan POX terlihat stabil saat dilakukan pengetesan ketika *background traffic* dinaikan menjadi 200 Mbps, sedangkan angka yang didapatkan Ryu terlihat stabil saat dilakukan pengetesan ketika *background traffic* berada di angka 100 Mbps. Secara keseluruhan, Ryu mengalami rata-rata *jitter* 2,94% lebih singkat dibandingkan POX dari keseluruhan ujicoba.

C. Throughput



GAMBAR 15
Grafik *throughput* pada kontroler POX

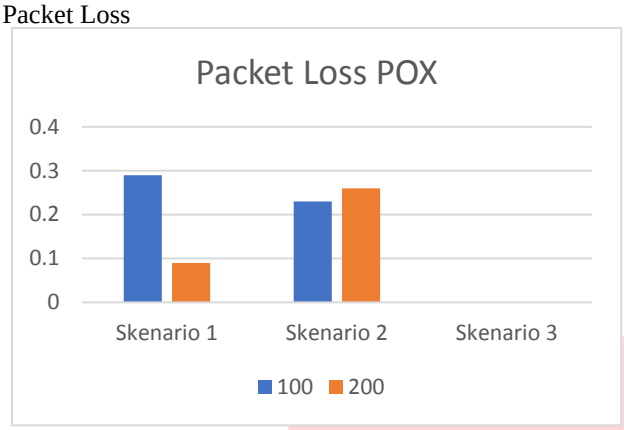


GAMBAR 16
Grafik *throughput* pada kontroler Ryu

Sama seperti *jitter*, kedua kontroler ini memiliki hasil yang saling mendekati. Dapat dilihat dari gambar

15 dan 16, kedua kontroler mampu menghasilkan angka *bitrate* di antara 34,5 Mbps s/d 38,4 Mbps, menunjukkan bahwa kedua kontroler mampu menjalankan tugasnya dengan baik. POX memiliki angka *bitrate* 3,57% lebih tinggi dibandingkan Ryu.

D. Packet Loss



GAMBAR 17
Grafik *packet loss* pada kontroler POX

Pada pengetesan yang terakhir, hanya terdapat data dari kontroler POX saja, dikarenakan kontroler Ryu tidak mengalami *packet loss* sama sekali dalam pengujiannya. Pada 2 skenario, yaitu skenario 1 dan 2, POX mengalami *packet loss*. Namun pada skenario ke 3, POX tidak mengalami *packet loss*.

V. KESIMPULAN

Setelah melakukan analisa pada data yang didapatkan dari beberapa percobaan di kontroler POX dan Ryu, maka hasil yang dapat disimpulkan adalah: Ryu memiliki angka *delay* dan *jitter* yang lebih rendah dibandingkan POX, sedangkan POX memiliki angka *throughput* yang lebih tinggi dibandingkan Ryu, serta Ryu tidak mengalami *packet loss* sama sekali selama percobaan dilakukan. Secara garis besar, masing-masing kontroler memiliki keunggulannya masing-masing.

REFERENSI

[1] E. Aza and J. Urrea, "Implementation of Round-Robin load balancing scheme in a wireless software defined network", in *2019 IEEE Colombian Conference on Communications and Computing (COLCOM)*, Barranquilla, 2019.

[2] K. Magade and A. Patankar, "Techniques for Load Balancing in Wireless LAN's", in *International Conference on Communication and Signal Processing*, Melmaruvathur, 2014.

[3] Y. Lin, C. Wang, Y. Lu, Y. Lai and H. Yang, "Two-tier dynamic load balancing in SDN-enabled Wi-Fi networks", *Wireless Networks*, vol. 24, no. 8, pp. 2811-2823, 2017. Available: 10.1007/s11276-017-1504-3.

[4] T. Nadeau and K. Gray, *SDN: software defined networks*, 1st ed. Beijing: O'Reilly, 2013.

[5] I. Haque and N. Abu-Ghazaleh, "Wireless Software Defined Networking: A Survey and Taxonomy", *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2713-2737, 2016. Available: 10.1109/comst.2016.2571118 [Accessed 10 February 2020].

[6] M. E. Mustafa, "Load Balancing Algorithms Round-Robin (RR), Least- Connection, and Least Loaded Efficiency," *Comput. Sci. Telecommun.*, vol. 51, no. 1, pp. 25–29, 2017.

[7] K. Salchow, "Load Balancing 101: The Evolution to Application Delivery Controllers," *F5 White Pap.*, pp. 1–7, 2007.

[8] N. Shah and M. Farik, "Static Load Balancing Algorithms In Cloud Computing: Challenges & Solutions," *Int. J. Sci. Technol. Res.*, vol. 4, no. 10, pp. 365–367, 2015.

[9] X. Foukas, M. K. Marina, and K. Kontovasilis, "Software Defined Networking Concepts," in *Software Defined Mobile Networks (SDMN): Beyond LTE Network Architecture*, First Edit., M. Liyanage, A. Gurtov, and M. Ylianttila, Eds. Chichester: Wiley Telecom, 2015, pp. 21–44.

[10] D. S. Rana, S. A. Dhondiyal, and S. K. Chamoli, "Software Defined Networking (SDN) Challenges, issues and Solution," *Int. J. Comput. Sci. Eng.*, vol. 7, no. 1, pp. 884–889, 2019, doi: 10.26438/ijcse/v7i1.884889.

[11] N. Shahzad, G. Mujtaba, and M. Elahi, "Benefits , Security and Issues in Software Defined Networking (SDN)," *NUST J. Eng. Sci.*, vol. 8, no. 1, pp. 38–43, 2015.

[12] P. A. Morreale and J. M. Anderson, *Software Defined Networking: Design and Deployment*. Boca Raton: CRC Press, 2015.

[13] S. Kaur, J. Singh, and N. S. Ghumman, "Network Programmability Using POX Controller," in *International Conference on Communication, Computing & Systems*, 2014, pp. 134–138, doi: 10.13140/RG.2.1.1950.6961.

[14] "Introduction to Mininet", *GitHub*. [Online]. Available: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet. [Accessed: 28- Feb- 2020].

[15] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6, doi: 10.1145/1868447.1868466.

[16] "noxrepo/pox", *GitHub*. [Online]. Available: https://github.com/noxrepo/pox. [Accessed: 30- Apr- 2020].

[17] "Getting Started – Ryu 4.34 documentation", *Ryu*. [Online]. Available: https://ryu.readthedocs.io/en/latest/getting_started.html. [Accessed: 15- Jul- 2022].