ANALISIS PERFORMA KONTROLER *RYU* DAN *PYRETIC* SEBAGAI KOMPONEN UTAMA PADA ARSITEKTUR *SOFTWARE DEFINED NETWORK*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Wicky Permadi Puji Asmara
NIM: 125150201111013

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2016

PENGESAHAN

ANALISIS PERFORMA KONTROLER *RYU* DAN *PYRETIC* SEBAGAI KOMPONEN UTAMA PADA ARSITEKTUR *SOFTWARE DEFINED NETWORK*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Wicky Permadi Puji Asmara
NIM: 125150201111013

Skripsi ini telah diuji dan dinyatakan lulus pada
24 November 2016
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Rakhmadhany Primananda, S.T, M.Kom	Sabriansyah Rizqika Akbar, S.T, M.Eng
NIK: 2016098604061001	NIP: 19820809 201212 1 004

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 24 November 2016

Wicky Permadi Puji Asmara

NIM: 125150201111013

KATA PENGANTAR

Puji syukur kami ucapkan kehadirat Tuhan Yang Maha Esa atas rakhmat dan hidyah-Nya sehingga kami dapat menyelesaikan skripsi yang berjudul "Analisis Performa Kontroler *Ryu* dan *Pyretic* Sebagai Komponen Utama Pada Arsitektur *Software Defined Network*".

Kami menyadari bahwa penulisan skripsi ini tidak akan terselesaikan tanpa bantuan dan saran yang sangat bermanfaat dari dosen pembimbing serta seluruh teman dan orang-orang yang terkait yang tanpa kenal lelah terus membantu. Oleh karena hal tersebut, kami mengucapkan terima kasih dengan setulus-tulusnya kepada seluruh pihak tersebut yang diantaranya :

1. Seluruh keluarga yang sangat saya sayangi, Barnoto dan Wahyu Arum yang telah memberi motivasi, kasih sayang, serta dukungan baik moril maupun materil. Adik-adik saya, Danang Pramudya Permadi dan Wisnu Aji Kusuma Permadi yang juga sangat saya sayangi.
2. Bapak Rakhmadhany Primananda, S.T, M.Kom. yang sangat saya hormati selaku dosen pembimbing I yang telah banyak memberikan ilmu, saran, dan motivasi untuk menyelesaikan skripsi ini.
3. Bapak Sabriansyah Rizqika Akbar, S.T, M.Eng yang sangat saya hormati selaku dosen pembimbing II yang telah banyak memberikan ilmu, saran, dan motivasi untuk menyelesaikan skripsi ini.
4. Seluruh civitas akademika Informatika Universitas Brawijaya terutama yang telah banyak membantu dan memberi dukungan selama penulisan skripsi ini
5. Teman-teman kos yang selalu menjadi teman seperjuangan sampai kapanpun. Fakhri, Iqbal, Mas Bahrur, Wawan, Ryan, Bagus, Faizal, dan seluruh teman lain yang tak bisa saya sebutkan seluruhnya.
6. Teman-teman seperjuangan skripsi jaringan, terima kasih telah memberikan semangat dan dukungan dalam pengerjaan skripsi ini.
7. Seluruh teman-teman PTIIK, khususnya Informatika 2012 terima kasih atas segala bantuan dan dukungannya.

Dengan segala kerendahan hati, dalam penyusunan skripsi ini saya menyadari masih banyak terdapat banyak kekurangan dan kesalahan, maka saya mengharapkan saran dan kritik yang bersifat membangun. Terima kasih, semoga laporan skripsi ini dapat bermanfaat bagi kita semua.

Malang, 24 November 2016

Penulis

ABSTRAK

Konsep jaringan *Software Defined Network* (SDN) adalah melakukan pemisahan antara *control plane* dan *data plane*. *data plane* berfungsi menentukan kemana paket akan diteruskan, sedangkan *control plane* mempunyai fungsionalitas infrastruktur jaringan seperti *routing* dan *switching*. *control plane* diimplementasikan dalam bentuk perangkat lunak.

Penelitian ini melakukan pengujian performa terhadap kontroler *Ryu* dan *Pyretic* dengan menggunakan Iperf yang bertujuan untuk mengetahui *throughput*, kapasitas *bandwidth*, mengetahui jumlah *jitter*, banyaknya *packet loss*, serta pemakaian CPU dan RAM pada jaringan SDN. Penelitian ini juga menggunakan algoritma *Non-weighted Dijkstra*.

Berdasarkan rata-rata total skenario, nilai rata-rata *throughput* kontroler *Pyretic* mencapai 619,83 MBps lebih tinggi dibanding kontroler *Ryu* yang mencapai 454,99 MBps. Nilai rata-rata *bandwidth* pada *Pyretic* mencapai 5,13 Gbps lebih tinggi dibanding *Ryu* yang mencapai 3,78 Gbps. Nilai rata-rata *jitter* pada kontroler *Pyretic* mencapai 0,020 ms lebih rendah dibanding kontroler *Ryu* yang mencapai 0,027 ms. Nilai rata-rata *paket loss Pyretic* mencapai 0,334 % lebih rendah dibanding *Ryu* yang mencapai 0,631 %. Rata-rata pemakaian CPU pada kontroler *Pyretic* lebih rendah dibanding kontroler *Ryu*. ketika *idle Pyretic* butuh rata-rata 8,3 %, sedangkan *Ryu* butuh rata-rata 13,3 %. Ketika dilakukan pengujian QoS, *Pyretic* butuh rata-rata 58,6 % sedangkan *Ryu* membutuhkan rata-rata 64,2 %. Dan rata-rata pemakaian RAM pada kontroler *Pyretic* mencapai 12,7 % lebih rendah dibanding *Ryu* yang mencapai 13,6 %. Hasil analisis pengujian tersebut menunjukkan bahwa kontroler *Pyretic* lebih unggul dibanding kontroler *Ryu* dalam seluruh paramater uji. Dalam penelitian ini, artinya kontroler yang paling tepat digunakan adalah *Pyretic*.

Kata kunci : *Software Defined Network, Ryu*, *Pyretic, Iperf, throughput, bandwidth, jitter, packet loss*, pemakaian *CPU,* performa kontroler

ABSTRACT

The concept of Software Defined Network network is the separation between the control plane and the data plane. Data plane serves to determine where the packet will be forwarded, and the control plane has network functionality such as routing and switching. The control plane is implemented in software.

This research measures the performance of the two controllers, Ryu and Pyretic. It uses Iperf to determine throughput, bandwidth capacity, determine the amount of jitter, packet loss percentage, also the amount of CPU and RAM usage on the SDN. This research also applies Non-weighted Dijkstra algorithm.

Based on the average of total scenario, the average Pyretic controller value of throughput reached 619,83 MBps, higher than Ryu controller which reached 454,99 MBps. The average of Pyretic bandwidth value reached 5,13 Gbps higher than the 3,78 Gbps of Ryu. The average value of jitter on the Pyretic controller reached 0,020 ms lower than Ryu controller which reached 0.027 ms. The average value of Pyretic packet loss reached 0.334 % lower than Ryu, which reached 0,631 %. The average of Pyretic CPU consumption is lower than the Ryu. when idle, Pyretic average reached 8,3 %, while Ryu average reached 13,3 %. When testing the QoS, Pyretic average reached 58,6 %, while Ryu average reached 64,2 %. And the average RAM usage on Pyretic reached 12,7 %, lower than Ryu which reached 13,6 %. The results of the analysis show that Pyretic controller is the better than Ryu controller in all QoS parameters test. In this research, it means that the most suitable controller to use is Pyretic.

Keywords: Software Defined Network, Ryu, Pyretic, Iperf, throughput, bandwidth, jitter, packet loss, CPU load, controller performance

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Beberapa tahun terakhir banyak inovasi dibidang teknologi komputer yang dikembangkan. Berbagai macam teknologi seperti *smartphone, notebook,* konsol *online games* dikembangkan dengan mengakses infrastruktur jaringan yang ada. Tetapi jaringan utama saat ini belum mengalami perubahan. Infrastruktur jaringan tersebut adalah jaringan tradisional yang menggunakan algoritma khusus pada perangkat jaringan seperti *router* dan *switch* untuk mengelola dan memelihara jaringan (Hu, 2014). Sekitar seperempat abad terakhir perangkat jaringan memiliki dua jenis pengolahan yaitu *data plane* dan *control plane* (Silver, 2013). Perangkat jaringan dapat mencapai 200, 2000 atau 20.000 dalam sebuah jaringan, artinya penggunaan *control plane* mencapai 20.000 dan harus selalu diperbarui (Silver, 2013). Dengan demikian diperlukan penanganan pada permasalahan jaringan tradisional tersebut.

Permasalahan jaringan tradisional yang telah disebutkan membutuhkan solusi yang adaptif terhadap perkembangan kebutuhan jaringan, sehingga solusi yang ditawarkan cenderung pada pengendalian data menggunakan perangkat lunak daripada meningkatkan sumber daya perangkat keras jaringan. Peneliti mewujudkan hal tersebut dengan paradigma *Software Defined Network* (SDN). Pada SDN, fungsionalitas infrastruktur jaringan seperti *routing* dan *switching* dikendalikan oleh perangkat lunak sehingga hal ini lebih fleksibel dan adaptif terhadap perkembangan kebutuhan jaringan dibandingkan mekanisme jaringan tradisional yang membutuhkan manajemen *console* pada ratusan atau ribuan perangkat (Underdahl, 2015).

Konsep jaringan *Software Defined Network* adalah melakukan pemisahan antara *control plane* dan *data plane*. *Data plane* berfungsi menentukan kemana paket akan diteruskan, sedangkan *control plane* menjalankan fungsionalitas infrastruktur jaringan seperti *routing* dan *switching*. *control plane* diimplementasikan dalam bentuk perangkat lunak (Silver, 2013). Implementasi *control plane* menggunakan kontroler. Namun pemilihan kontroler yang tepat menjadi masalah dalam penerapan jaringan tersebut.

Kontroler yang digunakan dalam penelitian ini adalah *Ryu* dan *Pyretic* karena menurut Rao pada tahun 2015, *Ryu* mendukung dua dari *use case* paling penting yaitu *service insertion and chaining,* dan *transport networks*. Sedangkan *Pyretic* memiliki persamaan terhadap *Ryu* dari segi bahasa pemrograman python, dukungan *rest api*, dan dukungan *platform* sendiri. Untuk mendapatkan hasil perbandingan kedua kontroler, penelitian menggunakan parameter QoS sebagai parameter uji. Hal tersebut dipilih karena QoS mengacu pada kemampuan jaringan untuk memberikan layanan yang baik pada lalu lintas jaringan dari berbagai teknologi (Caudhary, 2012). Hal itu menjadi fokus penelitian ini yaitu mengukur performa *throughput, bandwidth, jitter, dan packet loss* pada kontroler *Ryu* dan *Pyretic*. Dan untuk menambah parameter pengujian kontroler

ditambahkan parameter pemakaian CPU dan RAM untuk mengetahui pengaruh kontroler terhadap perangkat keras yang digunakan. Untuk menguji performa QoS serta pemakaian CPU dan RAM pada kedua kontroler tersebut diperlukan alat uji yang tepat. Iperf digunakan untuk menguji seluruh parameter QoS karena Iperf mendukung *tuning* dari berbagai parameter QoS yang terkait dengan *bandwidth, jitter,* dan *packet loss*. Untuk melakukan pengujian ini peneliti menggunakan simulator Mininet agar dapat menerapkan jaringan SDN tersebut. Dengan melakukan pengujian performa *QoS* serta pemakaian CPU dan RAM pada kontroler *Ryu* dan *Pyretic,* diharapkan penelitian ini dapat membantu pemilihan kontroler yang tepat.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan sebelumnya, maka dapat diambil perumusan masalah sebagai berikut:

1. Bagaimana menerapkan simulasi arsitektur jaringan SDN yang menggunakan kontroler *Ryu* dan *Pyretic*?
2. Bagaimana menganalisis performa kontroler *Ryu* dan *Pyretic* untuk parameter *throughput, bandwidth, jitter, packet loss* serta pemakaian CPU dan RAM menggunakan Iperf dalam simulasi jaringan SDN?
3. Bagaimanakah hasil perbandingan performa *throughput, bandwidth, jitter, packet loss* serta Pemakaian CPU dan RAM pada kontroler *Ryu* dan *Pyretic* setelah pengujian dilakukan?

1.3 Tujuan

Tujuan dilakukannya penelitian ini adalah :

1. Mengetahui penerapan simulasi arsitektur jaringan SDN yang menggunakan kontroler *Ryu* dan *Pyretic.*
2. Mengetahui perbandingan performa *throughput, bandwidth, jitter, packet loss,* pemakaian CPU dan RAM yang menggunakan simulasi Mininet.
3. Mengetahui hasil perbandingan *throughput, bandwidth, jitter, packet loss* serta pemakaian CPU dan RAM pada kontroler *Ryu* dan *Pyretic* dalam jaringan SDN.

1.4 Manfaat

Manfaat penelitian ini yaitu memberikan hasil perbandingan performa *throughput, bandwidth, jitter, packet loss* serta pemakaian CPU dan RAM dalam arsitektur jaringan SDN pada simulator Mininet, dengan membandingkan dua kontroler *Ryu* dan *Pyretic* yang mengimplementasikan satu topologi.

1.5 Batasan Masalah

Batasan masalah dalam penelitian ini adalah:

1. Implementasi topologi dan jaringan menggunakan simulasi Mininet.
2. Pengujian menggunakan aplikasi virtualisasi berupa Virtualbox.
3. Topologi yang diimplementasikan pada kedua Kontroler sama yaitu topologi dengan jumlah *host* 10 buah.
4. Alat yang digunakan untuk uji performa adalah IPerf.
5. Performa jaringan yang diukur adalah *throughput, bandwidth, jitter, packet loss,* serta Pemakaian CPU dan RAM.

1.6 Sistematika Pembahasan

Sistematika penulisan ditujukan untuk memberikan gambaran dan uraian dari pembuatan laporan skripsi secara garis besar yang meliputi beberapa bab sebagai berikut:

BAB I Pendahuluan

Bab ini berisi tentang latar belakang yang menjadi alasan pemilihan judul, perumusan masalah, tujuan penelitian, manfaat penelitian, pembatasan masalah, dan sistematika penelitian.

BAB II Landasan Kepustakaan

Bab ini menguraikan tentang konsep-konsep dan teori-teori yang berhubungan dengan permasalahan yang dirumuskan sebagai acuan dalam membahas permasalahan yang terjadi pada analisis performa kontroler *Ryu* dan *Pyretic*.

BAB III Metodologi

Bab ini menguraikan tentang metode dan langkah kerja yang dilakukan dalam penulisan tugas akhir yang terdiri dari studi literatur, perancangan perangkat lunak, implementasi perangkat lunak, pengujian dan analisis serta pengambilan kesimpulan dan saran. Dan Perancangan simulasi yang digunakan dalam penelitian.

BAB IV Hasil

Bab ini berisi tentang implementasi lingkungan pengujian jaringan SDN yang menggunakan dua Kontroler dalam Mininet. Bab ini juga membahas tenteng hasil dari pengujian yang telah dilakukan sesuai dengan implementasi lingkungan.

BAB V Pembahasan

Bab ini berisi mengenai pembahasan hasil pengujian dan analisis terhadap penerapan *Software-defined Network* yang diterapkan. Hasil pengujian

dipaparkan secara lengkap dalam bentuk perbandingan dan analisis dilakukan berdasar hasil pengujian yang telah dilakukan.

BAB VI Kesimpulan

Bab ini memuat kesimpulan yang diperoleh dari penerapan dan pengujian perangkat lunak yang dikembangkan dalam skripsi ini serta saran-saran untuk pengembangan lebih lanjut

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini akan dibahas tentang teori-teori yang terkait dengan penelitian yang terdiri dari tinjauan pustaka dan dasar teori. Tinjauan pustaka menjabarkan penelitian-penelitian yang sebelumnya telah ada atau yang diusulkan. Dan dasar teori menjabarkan seluruh teori pendukung penelitian yang meliputi konsep SDN dan parameter yang diuji.

2.1 Tinjauan Pustaka

Tinjauan pustaka membahas penelitian yang telah ada dan diusulkan. Pada penelitian ini kajian pustaka diambil dari beberapa penelitian yang pernah dilakukan. Dan kajian pustaka tersebut meliputi : Analisis Performansi Kontroler Floodlight Dan Beacon Sebagai Komponen Utama Pada Arsitektur *Software Defined Network* ditulis oleh Prima Nur Pratama (2015), *Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking* ditulis oleh Jehn-Ruey Jiang, dkk (2014), Pengujian Performa Kontroler *Software Defined Network* (SDN): POX dan Floodlight ditulis oleh Sawung Murdha Anggara (2015), dan *Comparisons of SDN OpenFlow Controllers over EstiNet: Ryu vs. NOX* ditulis oleh Shie-Yuan Wang, dkk. Berikut adalah persamaan dan perbedaan penelitian-penelitian tersebut seperti pada Tabel 2.1 :

Tabel 2. 1 Landasan Kepustakaan

No	Nama penelitian, Tahun dan Judul	Persamaan	Perbedaan	
			Penelitian terdahulu	Rencana Penelitian
1	Prima Nur Pratama . 2015. Analisis Performansi Kontroler *Floodlight* Dan *Beacon* Sebagai Komponen Utama Pada Arsitektur *Software Defined Network*	Menganalisis dua kontroler yang berbeda menggunakan alat simulasi Mininet	Kontroler yang digunakan adalah *Floodlight* dan *Beacon*. Alat uji dan topologi disimulasikan oleh *Cbench.* Parameter yang diuji adalah *flow setup rate* dan *flow setup delay* dan tidak dijelaskan aplikasi atau	Kontroler menggunakan *Ryu* dan *Pyretic*. Alat uji menggunakan *Iperf.* Parameter uji adalah *throughput, bandwidth, jitter, packet loss*, pemakaian CPU dan RAM. Algoritma yang diterapkan

			algoritma yang digunakan	adalah *Non-weighted Dijkstra*
2	Jehn-Ruey Jiang, dkk. 2014. *Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking*	Penelitian yang menggunakan topologi Abilene, alat simulasi Mininet, dan alat dan Iperf.	Menganalisis tiga algoritma berbeda yaitu *Extended Dijkstra's Shortest Path Algoritm*, *Non-weighted Dijkstra Algoritm* dan algoritma *Dijkstra s*tandar	Menganalisis performa QoS dan Pemakaian CPU dan RAM pada dua kontroler berbeda dengan algoritma *Non-weighted Dijkstra.*
3	Shie-Yuan Wang, dkk. 2015. *Comparisons of SDN OpenFlow Controllers over EstiNet: Ryu vs. NOX*	Membandingkan dua kontroler yang berbeda.	Kontroler yang digunakan adalah *Ryu* dan *Nox.* Dan juga menggunakan simulator dan emulator *Estinet OpenFlow*	Kontroler yang digunakan adalah *Ryu* dan *Pyretic.* Dan simulator menggunakan Mininet.
4	Sawung Murdha Anggara. 2015. Pengujian Performa Kontroler *Software Defined Network* (SDN): POX dan Floodlight	Menganalisis performa *QoS* dua kontroler berbeda.	Kontroler yang digunakan adalah *Pox* dan *Floodlight.* Alat uji dan topologi disimulasikan menggunakan simulator Cbench. Aplikasi yang digunakan adalah *forwarding tutorial_l2_hub*	Kontroler yang digunakan adalah *Ryu* dan *Pyretic.* Alat uji adalah Iperf dan simulator menggunakan Mininet. Aplikasi yang digunakan adalah *non-weighted Dijkstra*

2.2 Software Defined Network

Konsep jaringan *Software Defined Network* adalah melakukan pemisahan antara *control plane* dan *data plane*. *data plane* berfungsi menentukan kemana paket akan diteruskan, sedangkan *control plane* berfungsi logika jaringan seperti algoritma *routing*, SNMP, ekspor statistik *Netflow*, dan lainnya. *control plane* diimplementasikan dalam bentuk perangkat lunak (Silver, 2013)

control plane adalah bagian yang berfungsi menjalankan fungsionalitas infrastruktur jaringan seperti *routing* dan *switching*. *data plane* adalah bagian yang berfungsi untuk meneruskan paket-paket yang masuk ke suatu *port* pada perangkat *networking* menuju *port* keluar.

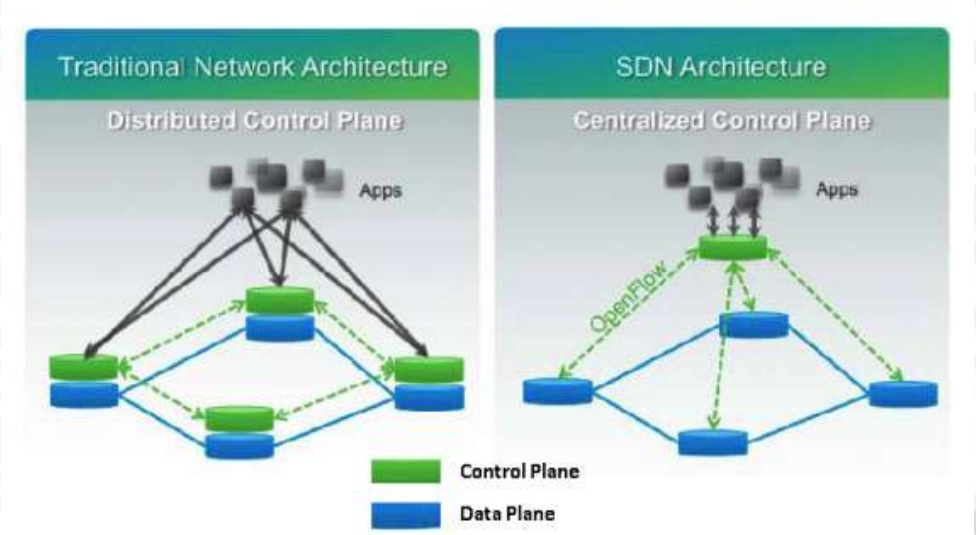
SDN hadir dalam menghadapi kompleksitas dan tantangan pada jaringan saat ini. Adapun tuntutan-tuntutan yang memberikan motivasi mengenai lahirnya SDN adalah sebagai berikut:

- Virtualisasi dan *cloud* : komponen dan entitas jaringan *hybrid* antara fisik *bare metal* dan virtual.
- *Orchestration* dan *scalability* : kemampuan untuk mengatur dan mengelola ribuan perangkat melalui sebuah *point of management*.
- *Programmability* dan *automation* : kemampuan untuk mengubah perilaku jaringan serta dapat melakukannya secara otomatis.
- Kinerja : kemampuan untuk memaksimalkan penggunaan perangkat jaringan, misalnya *load balancing*, *traffic engineering*, dan lain-lain (Pratama, 2015)

2.2.1 Perbandingan Konsep Jaringan Tradisional dan Jaringan SDN

Konsep jaringan SDN adalah melakukan pemisahan antara *control plane* dan *data plane*, dimana *data plane* tetap berada pada perangkat *Networking*, sedang *control plane* berada pada sebuah entitas terpisah bernama “Kontroler” yang akan menentukan perilaku jaringan dengan cara memungkinkan *data plane* untuk di program sehingga terbentuklah istilah *Software Defined Network* (SDN) yang mendefinisikan jaringan (Mendonca, 2013).

Sedangkan konsep jaringan tradisional memiliki perbedaan, perbedaaan tersebut adalah *control plane* dan *data plane* berada dalam satu perangkat *networking* yang sama. Artinya *control plane* dan *data plane* pada jarigan tradisional tidak dipisahkan seperti pada jaringan SDN. *control plane* pada jaringan tradisional merupakan struktur yang terdistribusi, yaitu setiap perangkat *networking* memiliki *control plane* tersendiri sehingga dapat membuat keputusan secara otonomi berdasarkan informasi yang dimilikinya, sedangkan *control plane* pada jaringan SDN memiliki struktur tersentralisasi. Dimana perangkat *networking* hanya memiliki kemampuan terbatas dalam membuat keputusan. Dan untuk membuat keputusan membutuhkan kehadiran kontroler (Pratama, 2015). Perbedaan antara jaringan tradisional dengan jaringan SDN dapat dilihat sesuai dengan gambar 2.1 :



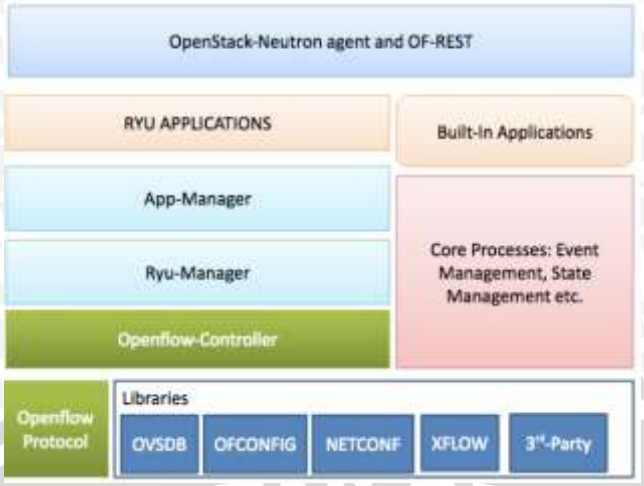
Gambar 2. 1 Konsep jaringan tradisional dan konsep jaringan SDN

Sumber : Cisco (2013)

Gambar 2.1 mengilustrasikan perbandingan antara konsep jaringan tradisional dengan konsep jaringan *Software Defined Network* (SDN). Gambar pada bagian kiri adalah konsep jaringan tradisional dimana *control plane* dan *data plane* berada pada tempat yang sama yaitu dalam perangkat *networking*. Sedangkan pada bagian gambar sebelah kanan merupakan konsep jaringan SDN yang menunjukkan penempatan *control plane* yang dipisah dengan *data plane*. (Mendonca, 2013)

2.2.2 Kontroler *Ryu*

Ryu adalah sebuah *framework* dari *Software Defined Network* (SDN) yang berbasis komponen yang diimplementasikan sepenuhnya dalam *Python*. Seperti kontroler SDN lainnya, *Ryu* juga menyediakan komponen perangkat lunak dengan API yang dapat dikembangkan oleh *developer* untuk membuat manajemen jaringan dan aplikasi kontrol baru. Tujuan dari *Ryu* adalah untuk mengembangkan sebuah sistem operasi untuk SDN yang memiliki kualitas cukup tinggi untuk digunakan dalam jaringan besar (Al-Somaidai, 2014). *Ryu* juga mendukung versi *OpenFlow* 1.0, 1.2, 1.3, 1.4, dan ekstensi Nicira. Arsitektur *Ryu* dapat dilihat pada gambar 2.1.



Gambar 2. 2 Arsitektur *Ryu*

Sumber : Rao (2014)

Arsitektur kontroler *Ryu* memiliki beberapa komponen penting. Komponen-komponen tersebut adalah sebagai berikut :

1. *Library*

   Ryu juga mendukung berbagai *library* yang dapat digunakan untuk berbagai operasi pemrosesan paket jaringan. *Ryu* mendukung *Open vSwitch Database Management Protocol* (OVSDB), *netconf*, *XFlow* (*Netflow* dan *sFlow*), sedangkan untuk *library* pihak ketiga dapat berupa *Open vSwitch Python binding.*

2. Protokol *OpenFlow* dan kontroller

   Ryu mendukung protokol *OpenFlow* sampai ke versi terbaru 1.4. Ini mencakup *library* protokol *OpenFlow* encoder dan decoder. Tabel 2.2 meragkum pesan *Ryu openflow protocol* dan struktur pesan. Tabel 2.2 adalah tabel yang dijabarkan oleh Rao tahun 2014 dalam judul *SDN Series Part Four: Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs*.

Tabel 2. 2 Rangkuman *Ryu openflow protocol messages*

Controller to Switch Messages	***Asynchronous Messages***	***Symmetric Messages***	**Structures**
Handshake, switch-config, flow-table-config, modify/read state, queue-config, packet-out, barrier, role-request	*Packet-in, flow-removed, port-status, and Error.*	*Hello, Echo-Request & Reply, Error, experimenter*	Flow-match

send_msg API and packet builder APIs	*set_ev_cls API and packet parser APIs*	*Both Send and Event APIs*	

Sumber : Rao (2014)

3. *Ryu Manager*

   Ryu manager adalah komponen utama pada *Ryu*. Ketika *Ryu manger* dijalankan, *Ryu manger* akan melakukan *listen* alamat IP tertentu atau misalnya 127.0.0.1 dan dengan port tertentu atau secara standar adalah 6633. Aplikasi ini adalah komponen dasar untuk semua aplikasi *Ryu.* Semua aplikasi turunan adalah *inherit* dari kelas *app manager Ryu*.

4. Aplikasi *Ryu*

   Ryu didistribusikan dengan beberapa aplikasi standar seperti *simple_switch, router, isolation, firewall*, *GRE tunel*, topologi, dan VLAN. Aplikasi *Ryu* merupakan entitas yang melaksanakan berbagai fungsi meliputi mengirim *asynchronous event* antar aplikasi yang digunakan (Rao, 2014).

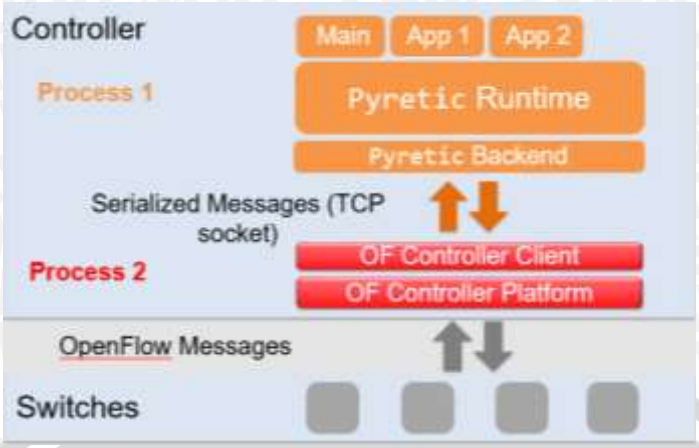
2.2.3 Kontroler *Pyretic*

Pyretic diperkenalkan sebagai bahasa pemrograman *Software Defined netwok* (Reich, 2013). Hal ini memungkinkan *programmer* untuk berfokus pada bagaimana menentukan *policy* jaringan. *Policy* ini bertindak sebagai sebuah fungsi atau *method* (Reich, 2013). *Programmer Pyretic* juga dapat membuat *dynamic policies* yang perilakunya akan berubah sewaktu-waktu. Singkatnya, *Pyretic* memungkinkan programmer SDN untuk membuat aplikasi jaringan modular (Jiang, 2014). Beberapa *policy* yang dapat digunakan dalam *Pyretic* ditunjukkan pada tabel 2.3.

Tabel 2. 3 *Policy* pada *Pyretic*

Syntax	Keterangan
identify	Mengembalikan paket asli
drop	Mengembalikan set kosong
match(f=v)	Identifikasi f = v, jika tidak *drop*
modify(f=v)	Mengembalikan paket dengan f = v
fwd(a)	*Modify*(port=a)
flood()	Mengembalikan satu paket untuk setiap *local port* dalam *spannig tree*

Sumber : Reich (2013)



Gambar 2. 3 Arsitektur *Pyretic*

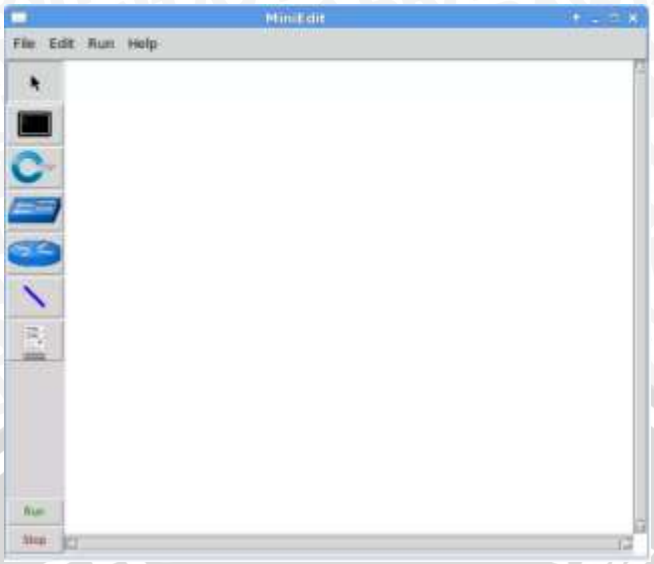
Sumber : Reich (2013)

Kontroler *Pyretic* didukung protokol *OpenFlow* versi 1.0 yang berjalan untuk komunikasi perangkat kontroler seperti ditunjukkan pada gambar 2.3. *Pyretic* juga mendukung aplikasi seperti *mac learning* yang dapat digunakan langsung pada modul di dalam direktori *modules.*

2.3 Emulator Mininet

Mininet adalah sebuah emulator jaringan *open source* yang mendukung protokol *OpenFlow* untuk arsitektur *SDN*. Mininet menggunakan pendekatan virtualisasi untuk membuat jaringan *host* virtual, *switch,* kontroler, dan *link*. Pada Mininet sudah terdapat beberapa topologi bawaan yang dapat langsung digunakan dengan menggunakan perintah tertentu. Beberapa topologi bawaan tersebut antara lain topologi *single*, *tree* dan *linear*. Selain topologi yang telah tersedia tersebut, Mininet dapat menjalankan topologi *custom* sesuai dengan topologi yang diinginkan dengan menambahkan *script* berbahasa *phyton* (Pratama, 2015).

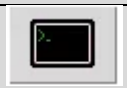
Mininet tidak hanya berbentuk aplikasi CLI, namun Mininet juga mendukung bentuk aplikasi berbasis GUI. Aplikasi basis GUI ini bernama MiniEdit. Aplikasi berbasis GUI ini juga diimplementasikan dengan menggunakan *python*. Mininet memiliki antarmuka yang sederhana seperti pada gambar 2.4.



Gambar 2. 4 Antarmuka Mininet yang berbasis GUI (MiniEdit)

Dalam miniedit terdapat komponen yang dapat digunakan untuk membuat topologi jaringan. Berikut adalah penjelasan komponen pada miniedit seperti pada tabel 2.4.

Tabel 2. 4 Fungsi komponen kontroler Mininet

No	Gambar	Nama Komponen	Fungsi
1		*Host*	Membuat node yang berfungsi sebagai *host* komputer
2		*Switch Tool*	Membuat *OpenFlow-enabled*
3		*Legacy Switch*	Membuat *learning ethernet switch*
4		*Legacy Router*	Membuat router independen
5		*NetLink*	Menghubungkan antar node
6		*Controller*	Membuat sebuah kontroler

2.4 Algoritma *Non-weighted Dijkstra*

Algoritma *Non-weighted Dijkstra* adalah algoritma *Dijkstra* yang tanpa menggunakan bobot atau *weight* dalam pengaplikasiannya, atau penggunaan nilai pada *edge* bernilai 1. Algortima Dijkstra adalah algritma pencarian jarak terpendek untuk sebuah graf terarah dengan bobot-bobot sisi yang bernilai bukan negatif.

Artinya Algortima *Non-weighted Dijkstra* mencari jalur dengan menghitung jumlah hop atau pasangan node awal dan node tujuan (Jiang, 2014). *Pseudocode* pada algoritma *Non-weighted Dijkstra* ini dapat dilihat seperti pada tabel 2.5.

Tabel 2. 5 *Non-weighted Dijkstra Pseudocode*

```
1   function Dijkstra(Graph, source):
2       create vertex set Q
3       for each vertex v in Graph:
4           dist[v] ← INFINITY
5           prev[v] ← UNDEFINED
6           add v to Q
7       dist[source] ← 0
8       while Q is not empty:
9           u ← vertex in Q with min dist[u]
10          remove u from Q
11          for each neighbor v of u:
12              if dist[u] + 1 < dist[v]:
13              dist[v] ← dist[u] + 1
14              prev[v] ← u
15       return dist[], prev[]
```

Berdasarkan tabel 2.5, Q adalah himpunan dari *vertex.* Dist[v] menggambarkan jarak vertex tujuan sedangkan prev[v] menggambarkan vertex sebelum berpindah ke vertex tujuan.

2.5 Alat Uji IPerf

IPerf atau Iperf3 adalah alat untuk pengukuran aktif pada *bandwidth* maksimum yang dapat dicapai pada jaringan *IP*. Mendukung *tuning* dari berbagai parameter yang terkait dengan *timing, buffer*, dan protokol (*TCP,UDP,SCTP IPv4* dan *IPv6*). Iperf dapat digunakan untuk mengukur *bandwidth, packet loss*, dan parameter lainnya. IPerf pada awalnya dikembangkan oleh NLANR atau DAST. Iperf3 pada prinsipnya dikembangkan oleh ESNet atau Lawrence Berkelev National.

Iperf memiliki banyak opsi perintah yang dapat dijalankan sesuai dengan skenario suatu pengujian. Opsi perintah pada Iperf dibagi menjadi tiga yaitu opsi perintah secara umum yang dapat dieksekusi oleh *server* maupun *client* seperti pada Tabel 2.6, opsi perintah untuk *server* seperti pada Tabel 2.7, dan opsi perintah untuk *client* seperti pada Tabel 2.8.

Tabel 2. 6 Opsi perintah umum pada Iperf

Perintah Umum Iperf	
Command Line	**Deskripsi**
p, --port ***n***	Port yang digunakan untuk koneksi antara *client* dan *server*. Pengaturan awal port adalah 5201.
--cport *n*	Pilihan untuk port *client*-side secara spesifik.
-f, --format***[kmKM]***	Untuk menspesifikasikan format tampilan *bandwidth*. Meliputi : 'k' = Kbits/sec 'K' = KBytes/sec 'm' = Mbits/sec 'M' = MBytes/sec
-i, --interval ***n***	Untuk mengatur tampilan interval waktu yang dinginkan
-F, --file name	***client*:** read from the file and write to the network, instead of using random data; ***server*:** read from the network and write to the file, instead of throwing the data away.

Tabel 2. 7 Opsi perintah *server* pada Iperf

Perintah untuk *server* Iperf	
Command Line	**Deskripsi**
-s, --*server*	Menjalankan iperf sebagai *server*
-D, --daemon	menjalankan *server* sebagai daemon.
-I, --pidfile***file***	membuat file dengan ID proses, sangat berguna ketika sebagai daemon

Tabel 2. 8 Opsi perintah *client* pada Iperf

Perintah untuk *client* Iperf	
Command Line	**Deskripsi**
-c, --*client* ***host***	Menjalankan iperf sebagai *client*
-u, --udp	Menggunakan *client* sebagai UDP
-b, --*bandwidth* ***[KM]***	Mengatur batas *bandwidth*. Batas standar 1 Mbit/sec untuk UDP, tak terhingga untuk TCP
-t, --time ***n***	Mengatur waktu transmisi. Pengatran standar 10 detik.
-P, --parallel ***n***	Jumlah koneksi simultan ke *server*. Jumlah standar adalah 1.
-w, --windo ***[KM]***	Mengatur ukuran *socket buffer*. Jika TCP, digunakan sebagai ukuran *TCP window*.

iPerf dapat melakukan pengujian terhadap dua protokol. Secara sederhana dapat disimpulkan seperti berikut:

1. Pengujian terhadap potokol UDP : Ketika digunakan untuk menguji kapasitas UDP, iPerf memungkinkan pengguna untuk memberikan hasil *packet loss* dan *jitter*.
2. Pengujian terhadap potokol TCP : Ketika digunakan untuk menguji kapasitas TCP, iPerf mengukur *throughput* dan *bandwidth.*

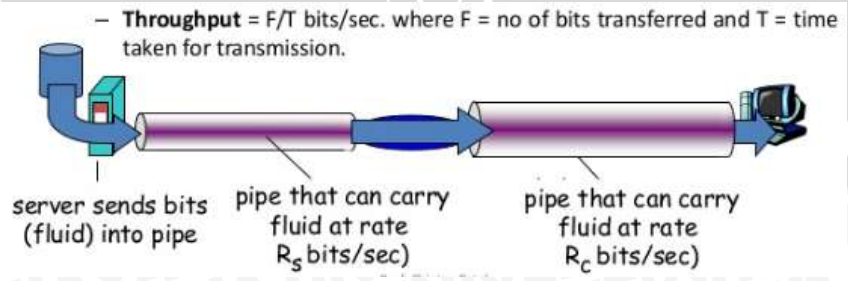
2.6 Quality of Service (QoS)

Quality of Service mengacu pada sistem telekomunikasi untuk memberikan layanan yang lebih baik untuk jalur yang dipilih melalui jaringan yang heterogen. *Quality of Service* adalah kemampuan suatu jaringan untuk menyediakan layanan yang lebih baik pada trafik data pada berbagai jenis platform teknologi. QoS dapat mempresentasikan tipe kebutuhan untuk memberikan garansi kepada pengguna, seberapa lama pengguna atau penerima menunggu, dan seberapa banyak data yang hilang. Komponen-komponen dari QoS menurut (Ferguson & Huston. 1998) adalah *throughput, jitter, delay,* dan *packet loss* (Wibowo, 2014).

2.6.1 Throughput

Throughput adalah kecepatan transfer data secara efektif yang diukur dalam satuan *bits per second* (Mbps) dan jumlah paket yang sukses tiba di alamat tujuan selama interval waktu tertentu dibagi dengan durasi interval waktu. Ilustrasi *throughput* dapat dilihat pada gambar 2.5 (Ross, 2013). Untuk menghitung rata-rata *throughput* dapat dilakukan dengan menggunakan rumus seperti berikut:

$$Throughput = \frac{Ukuran\ file\ (dalam\ satuan\ bps)}{waktu\ (dalam\ satuan\ second)} = \frac{Transfer\ (bps)}{interval\ (s)}$$



Gambar 2. 5 Ilustrasi *throughput*

Sumber: Ross (2013)

2.6.2 Bandwidth

Bandwidth adalah nilai hitung atau perhitungan konsumsi transfer data telekomunikasi yang terjadi antara komputer *server* dan komputer *client* dalam waktu tertentu dalam sebuah jaringan komputer. *Bandwidth* juga bisa berarti jumlah konsumsi paket data per satuan waktu dinyatakan dengan *bit* per *second* (bps) (Wibowo, 2014).

2.6.3 Jitter

Jitter merupakan variasi delay antar paket yang terjadi pada jaringan berbasis IP. Besarnya nilai *jitter* akan sangat dipengaruhi oleh variasi beban trafik dan besarnya tumbukan antar paket (*congestion*) yang ada dalam jaringan tersebut. Semakin besar beban trafik di dalam jaringan akan menyebabkan semakin besar pula peluang terjadinya *congestion*, dengan demikian nilai *jitter*nya akan semakin besar. Semakin besar nilai *jitter* akan menyebabkan nilai QoS semakin turun. Kategori kinerja jaringan berbasis IP dalam *jitter* versi *Telecommunications and Internet Protocol Harmonization Over Networks* (TIPHON) mengelompokkan menjadi empat kategori penurunan kinerja jaringan berdasarkan nilai *jitter* seperti terlihat pada tabel 2.9 (Wibowo, 2014).

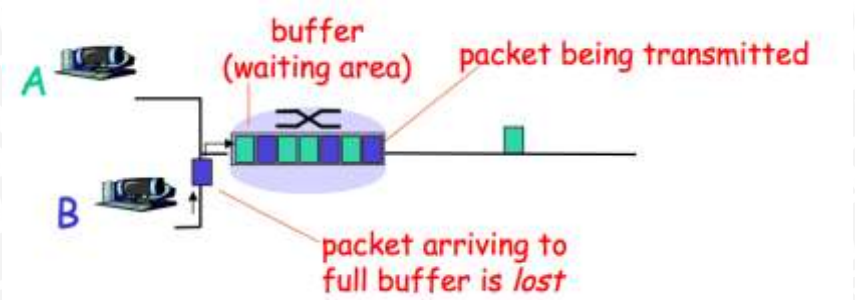
Tabel 2. 9 Kategori kinerja jaringan dalam *jitter*

Kategori degradasi	*Jitter*
Sangat Bagus	0 ms
Bagus	75 ms
Sedang	125 ms
Buruk	225 ms

Sumber : ETSI (1999)

2.6.4 Packet loss

Packet loss merupakan parameter yang menggambarkan kondisi yang menunjukkan jumlah paket yang hilang. *Packet loss* dapat disebabkan karena terputusnya jalur antara alamat sumber dan alamat tujuan. *Router* dapat membuang paket jika tidak ada tempat untuk menyimpan paket seperti Gambar 2.6 Hal tersebut dapat terjadi jika lalu lintas padat dan paket datang ketika antrian sedang penuh (Ross, 2013).



Gambar 2. 6 Ilustrasi *packet loss*

Sumber: Ross (2013)

Dan Kategori kinerja jaringan berbasis IP dalam *packet loss* menurut TIPHON dapat dikategorikan seperti pada tabel 2.10 sebagai berikut :

Tabel 2. 10 Kategori kinerja jaringan dalam *packet loss*

Kategori degradasi	*Packet loss*
Sangat Bagus	0 %
Bagus	3 %
Sedang	15 %
Buruk	25 %

Sumber : ETSI (1999)

Untuk menghitung *packet loss* dapat menggunakan persamaan sebagai berikut :

$$Packet\ loss = \frac{(paket\ data\ terkirim - paket\ data\ diterima)}{paket\ data\ yang\ dikirim} \times 100\%$$

2.7 Pemakaian CPU dan RAM

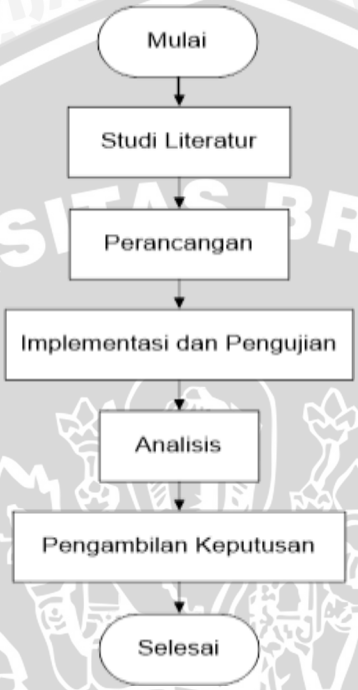
Pemakaian CPU atau biasa disebut *CPU load (CPU usage)* adalah jumlah penggunaan prosesor untuk suatu kegiatan pemrosesan instruksi, program komputer atau sistem operasi yang dinyatakan dalam bentuk persentase. Penggunaan CPU menggambarkan persentase waktu yang prosesor telah dihabiskan untuk aplikasi tertentu. Bahkan ketika penggunaan CPU pada 0 persen, CPU terus melakukan tugas ringan, seperti menanggapi gerakan *keyboard* dan *mouse* (Seehorn, 2016).

RAM atau *Random Access Memory* ditemukan dalam modul yang terhubung ke *motherboard*. Setiap aplikasi yang terbuka menggunakan sejumlah RAM. RAM bukan memori penyimpanan, tetapi hanya memori yang diperlukan untuk benar-benar menjalankan program. Pemakaian RAM adalah Prosentase beban RAM yang digunakan oleh aplikasi yang dijalankan. (Seehorn, 2016)

BAB 3 METODOLOGI

3.1 Metode Penelitian

Bab ini menjelaskan tentang langkah-langkah yang dilakukan dalam penelitian. Langkah-langkah dalam penelitian dapat digambarkan dalam bentuk diagram alir seperti pada Gambar 3.1 berikut :



Gambar 3. 1 Diagram alir metode penelitian

Berikut ini adalah penjelasan diagram alir tahapan penelitian :

1. Mulai,
2. Melakukan studi literatur yang berkaitan dengan topik permasalahan dan rumusan masalah yang diangkat dalam penelitian,
3. Merancang kebutuhan, topologi, simulasi jaringan SDN yang akan digunakan dalam pengujian,
4. Mengimplementasikan simulasi jaringan sesuai dengan rancangan, melakukan pengujian dan memaparkan hasil pengujian,
5. Melakukan analisis terhadap hasil pengujian yang telah dilakukan,
6. Menyimpulkan hasil pengujian dan memberikan saran untuk penelitian selanjutnya,
7. Selesai.

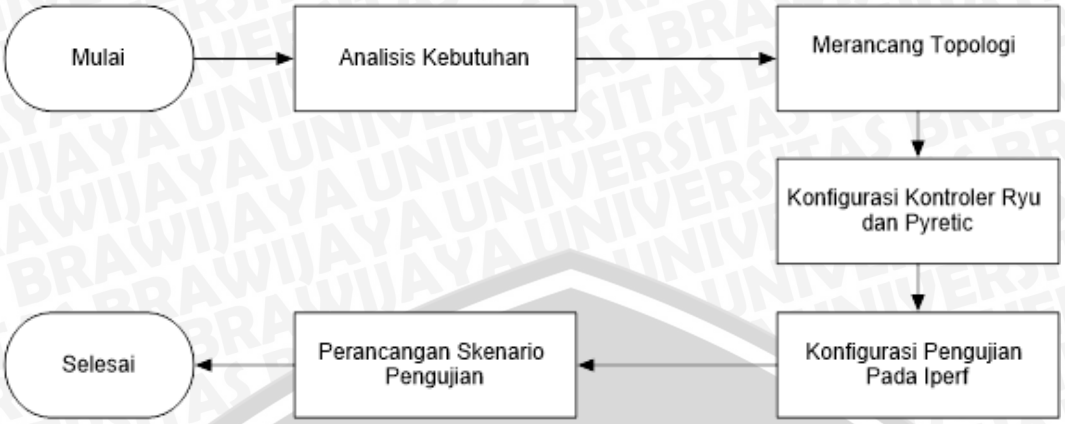
3.1.1 Studi Literatur

Pada bagian ini membahas dasar teori pendukung analisis seluruh kebutuhan untuk membangun analisis dua kontroler pada SDN yang mengimplementasikan topologi 10 *host* dalam Mininet. Adapun yang dijadikan dalam bahan studi literatur adalah dasar teori untuk dapat menganalisis dan merancang sistem meliputi :

1. SDN
2. Kontroler yang digunakan dalam SDN
   a. Kontroler *Ryu*
   b. Kontroler *Pyretic*
3. Topologi Jaringan
4. Simulator Mininet
5. Bahasa Pemrograman
   a. Python
6. Algoritma *Non-weighted Dijkstra*
7. Paramater Uji
   a. *Throughput*
   b. *Bandwidth*
   c. *Jitter*
   d. *Packet loss*
   e. Pemakaian CPU dan RAM
8. Alat uji Iperf

3.1.2 Perancangan

Perancangan dilakukan agar dapat memudahkan langkah implementasi dan pengujian penelitian. Tahapan perancangan meliputi analisis kebutuhan perangkat lunak dan perangkat keras, rancangan topologi, konfigurasi kontroler *Ryu* dan *Pyretic*, dan pengujian menggunakan Iperf. Tahapan perancangan dapat dilihat seperti pada Gambar 3.2.



Gambar 3. 2 Diagram alir tahapan perancangan sistem

Berikut ini adalah penjelasan diagram alir tahapan perancangan sistem :

1. Mulai perancangan,
2. Analisis kebutuhan merangkum dari analisis kebutuhan perangkat lunak dan perangkat keras yang digunakan untuk pengujian,
3. Perancangan topologi untuk diterapkan pada penelitian. Tahapan ini juga meliputi konfigurasi alamat IP dan *port* tiap kontroler pada topologi,
4. Konfigurasi kontroler *Ryu* dan *Pyretic* pada tahapan ini meliputi implementasi algoritma *Non-weighted Dijkstra* yang diimplementasikan pada kedua kontroler,
5. Konfigurasi pengujian pada iperf meliputi opsi perintah yang digunakan dalam pengujian, banyaknya pengujian, parameter yang diuji dan pemilihan *host* uji pada topologi menggunakan Iperf,
6. Melakukan perancangan skenario pengujian pada setiap parameter uji,
7. Selesai perancangan.

3.1.3 Implementasi dan Pengujian

Tahap ini merupakan tahap implementasi dan pengujian. Tahap Implementasi merupakan penjelasan secara umum penerapan seluruh perancangan yang telah ditentukan. Implementasi dilakukan dengan mengacu pada perancangan sistem. Implementasi meliputi:

1. Mengimplementasikan topologi pada Mininet
2. Mengimplementasikan dua kontroler *Ryu* dan *Pyretic* dengan melakukan implementasi algoritma *Non-weighted Dijkstra.* Menggunakan alamat IP 127.0.0.1 dan alamat *port* 6633.
3. Menjalankan simulasi Mininet dan kontroler
4. Menjalankan Iperf untuk pengujian dalam topologi

Pengujian dilakukan setelah implementasi telah selesai. Terdapat beberapa skenario dalam pengujian, meliputi pengujian *throughput, bandwidth, jitter,*

packet loss serta pemakaian CPU dan RAM. Pengujian seluruh parameter dilakukan dengan secara lima kali setiap pengujian, hasilnya akan dilakukan pendataan rata-rata. Hal tersebut dilakukan agar hasil data yang dapat digunakan menjadi valid. Pada akhir tahap ini akan diperoleh hasil pengujian yang sudah terdokumentasi.

3.1.4 Analisis

Analisis akan dilakukan setelah seluruh pengujian terhadap dua kontroler telah selesai. Analisis dilakukan dengan membandingkan hasil yang diperoleh ke dalam bentuk tabel dan grafik. Setiap tabel dan grafik mempresentasikan hasil pengujian secara lengkap. Analisis ini mencakup lima parameter yang diuji, yaitu *throughput, bandwidth, jitter, packet loss* serta pemakaian CPU dan RAM. Analisis memberikan penjelasan hasil secara detil dan secara menyeluruh.

3.1.5 Pengambilan Keputusan

Pengambilan keputusan dilakukan setelah semua tahapan perancangan, implementasi, dan pengujian sistem yang dibangun telah selesai dilakukan. Kesimpulan diambil dari hasil analisis lima parameter pengujian. Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memberikan pertimbangan penelitian lebih lanjut.

3.2 Perancangan

Perancangan dilakukan agar dapat memudahkan langkah implementasi dan pengujian penelitian. Tahapan perancangan meliputi analisis kebutuhan perangkat lunak dan perangkat keras, rancangan topologi, konfigurasi kontroler *Ryu* dan *Pyretic*, perancangan skenario pengujian, dan pengujian menggunakan Iperf. Perancangan ini akan diimplementasikan dalam pengujian.

3.2.1 Analisis Kebutuhan

Analisis kebutuhan bertujuan untuk membahas bagaimana sistem harus menyelesaikan masalah pada sistem yang akan dibangun atau dalam hal ini adalah jaringan SDN. Selain itu dalam sub bab ini akan diberikan penjelasan mengenai penggunaan Mininet sebagai simulator yang digunakan pada penelitian. Oleh karena itu analisis kebutuhan ini untuk menggambarkan kebutuhan kebutuhan dari sistem agar dapat bekerja sesuai dengan yang diharapkan.

3.2.1.1 Perangkat Lunak Yang Digunakan

Penelitian ini menggunakan perangkat lunak yang bersifat *free* dan *open source*. Kedua kontroler yang digunakan yaitu *Pyretic* dan *Ryu* dibangun menggunakan bahasa pemrograman Python yang juga bersifat *open source*. Penggunaan perangkat lunak yang lain berupa simulator Mininet untuk membangun topologi jaringan SDN dan Alat penguji IPerf. Sedangkan sistem operasi yang digunakan untuk penelitian ini adalah Xubuntu 14.04 LTS 64 bit. Seluruh program dan aplikasi menggunakan perangkat lunak *open source*. Untuk

memudahkan berikut adalah daftar perangkat lunak yang digunakan penelitian seperti pada tabel 3.1.

Tabel 3. 1 Perangkat lunak yang digunakan

No	Perangkat Lunak	Fungsi
1	Mininet	simulator jaringan SDN
2	*Pyretic* dan *Ryu*	kontroler yang digunakan pada penelitian
3	IPerf	Alat untuk menguji performa kontroler
4	Xubuntu 14.04 64 bit	Sistem operasi yang digunakan untuk penelitian SDN ini

3.2.1.2 Perangkat Keras Yang digunakan

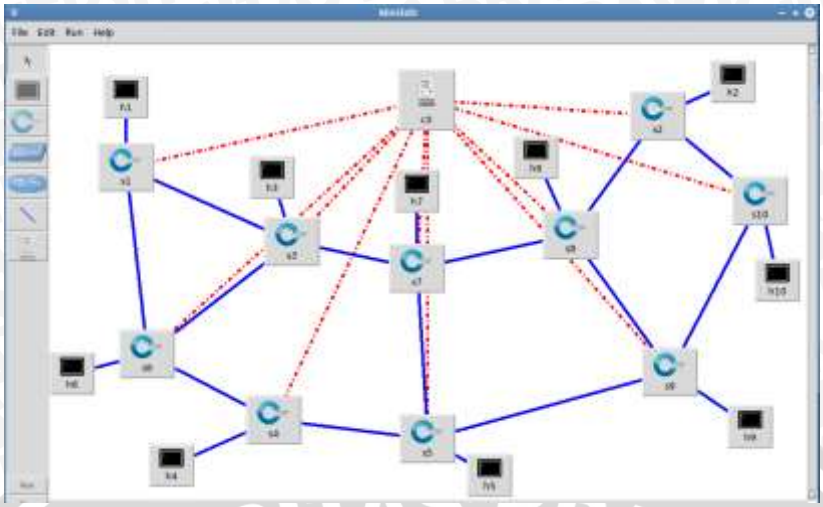
Penelitian ini menggunakan sebuah laptop untuk melakukan pengujian penelitian ini. Kontroler *Pyretic* dan *Ryu* dipasang pada satu laptop yang sama dan digunakan secara bergantian. Perangkat keras dalam pengujian dapat berupa PC desktop maupun laptop. Penelitian ini juga menggunakan virtualbox yang dipasang pada laptop. Berikut merupakan spesifikasi laptop yang digunakan sebagai alat untuk membangun pengujian kontroler seperti pada tabel 3.2 :

Tabel 3. 2 Perangkat keras yang digunakan

No	Perangkat Keras	Spesifikasi
1	CPU	Intel Core i3-4030U, 1.9 Ghz (Vbox : 2 core)
2	RAM	DDR3L 6 GB (Vbox : 2674 MB)
3	HDD	500 GB (Vbox : 20 GB)

3.2.2 Merancang Topologi

Pengujian performa kedua kontroler menggunakan mininet. Agar minnet dapat berjalan dengan baik diperlukan sebuah topologi. Penelitian ini menggunakan topologi yang sama pada kedua kontroler agar mendapatkan hasil yang akurat. Topologi yang digunakan adalah topologi yang menggunakan 10 host dan 10 buah *switch* yang dihubungkan. 10 host yang digunakan akan dibagi menjadi 1 *host* sebagai *server* dan 9 *host* menjadi *client* seperti pada gambar 3.4.



Gambar 3. 3 Topologi yang dibuat menggunakan miniedit.py

agar mudah dipahami, konfigurasi pada topologi dirangkum seperti pada tabel 3.3 berikut ini :

Tabel 3. 3 Konfigurasi topologi

Parameter	Konfigurasi
Jumlah Switch	10
Jumlah *Host*	10
Jumlah kontroler	1
Alamat IP kontroler	127.0.0.1
Port yang digunakan	6633 (*Ryu* & *Pyretic*)
Tipe kontroler	*Remote Controller*

Tabel 3. 4 Alamat IP *host* pada topologi

N *Host*	1	2	3	4	5	6
Alamat IP	10.0.0.1	10.0.0.2	10.0.0.3	10.0.0.4	10.0.0.5	10.0.0.6
N *Host*	7	8	9	10		
Alamat IP	10.0.0.7	10.0.0.8	10.0.0.9	10.0.0.10		

Pada gambar 3.4 yang merupakan topologi yang dibuat menggunakan aplikasi miniedit.py. Topologi yang digunakan adalah topolog yang terdiri dari 10 *switch* yang berlabel s1 hingga s10 dan 10 *host* yang berlabel h1 sampai h10, dan kontroler yang berlabel c0. Masing-masing akan saling terhubung sesuai dengan gambar 3.4. Alamat IP dan *port* yang digunakan kedua kontroler sama yaitu 127.0.0.1 pada port 6633 karena simulasi dilakukan secara lokal. Dan alamat IP pada tiap *host* dapat dilihat seperti pada tabel 3.4.

3.2.3 Konfigurasi kontroler *Ryu* dan *Pyretic*

Pada tahap ini, konfigurasi pada kedua kontroler ditentukan agar dapat digunakan pada taham implementasi dan pengujian. Konfigurasi tersebut meliputi tipe yang digunakan kontroler, alamat IP, *port,* dan aplikasi yang digunakan masing-masing kontroler. Dan konfigurasi pada kontroler *Ryu* dapat diketahui seperti pada tabel 3.5 berikut :

Tabel 3. 5 Konfigurasi kontroler *Ryu*

Parameter	Konfigurasi
Tipe Kontroler	*Remote Controller*
Alamat IP	127.0.0.1
Port yang digunakan	6633
Aplikasi (Algoritma)	*Non-weighted Dijkstra*

Dan konfigurasi pada kontroler *Pyretic* dapat diketahui seperti pada tabel 3.6 berikut :

Tabel 3. 6 Konfigurasi kontroler *Pyretic*

Parameter	Konfigurasi
Tipe Kontroler	*Remote Controller*
Alamat IP	127.0.0.1
Port yang digunakan	6633
Aplikasi (Algoritma)	*Non-weighted Dijkstra*

alamat IP dan *port* yang digunakan kedua kontroler sama yaitu 127.0.0.1 pada port 6633 karena simulasi dilakukan secara bergantian dan menggunakan jaringan *localhost*.

3.2.4 Konfigurasi Pengujian Menggunakan Iperf

Pada tahap ini rancangan pengujian menggunakan aplikasi Iperf. Iperf ini digunakan untuk menguji parameter QoS yaitu *throughput, bandwidth, jitter,* dan *packet loss*. Sedangkan rancangan pengujian pemakaian CPU dilakukan monitoring terhadap persentase CPU ketika melakukan pengujian sesuai dengan skenario. Konfigurasi pengujian dapat dilihat seperti pada tabel 3.7.

Tabel 3. 7 Opsi perintah Iperf dalam pengujian QoS

Parameter Uji	Opsi Perintah Iperf	
	Server	***Client***
throughput	Iperf –s	Iperf –c 10.0.0.1 –f m Iperf –c 10.0.0.1 –w <ukuran window> -f m
bandwidth	Iperf –s	Iperf –c 10.0.0.1 –f m Iperf –c 10.0.0.1 –w <ukuran window> -f m
Jitter	Iperf –s –u	Iperf –c 10.0.0.1 –b <batas *bandwidth*>
Packet loss	Iperf –s –u	Iperf –c 10.0.0.1 –b <batas *bandwidth*>

Pengujian pemakaian CPU tidak masuk dalam opsi perintah Iperf karena pengujian dilakukan dengan cara merekam aktivitas CPU yang sedang melakukan pengujian terhadap parameter. Dan aktivitas CPU yang direkam adalah pengujian terhadap parameter *throughput*.

3.2.5 Perancangan Skenario Pengujian

Perancangan skenario pengujian dilakukan untuk menentukan urutan dan implementasi pengujian. Dengan menggunakan skenario pengujian, data hasil pengujian akan didapatkan dengan maksimal. Skenario pengujian dibagi menjadi tiga tahap dan memiliki beberapa skenario yang berbeda pada tiap tahap. Tahapan-tahapan tersebut yakni:

1. **Tahapan pertama** :

   Pengujian terhadap parameter *throughput* dan *bandwidth* menggunakan protokol TCP. Pengujian ini menggunakan opsi perintah Iperf sesuai dengan konfigurasi Iperf pada pengujian. Tahap ini memiliki tiga skenario dan masing masing dilakukan sebanyak lima kali pengujian. Dan tiga skenario tersebut adalah :

   a. **Skenario 1** : Pengujian *throughput* dan *bandwidth* dengan opsi perintah Iperf dengan ukuran *window* standar yaitu 85,3 KB.

   b. **Skenario 2** : Pengujian *throughput* dan *bandwidth* dengan opsi perintah Iperf dengan ukuran *window* lebih besar dari standar yaitu 170 KB.

   c. **Skenario 3** : Pengujian *throughput* dan *bandwidth* dengan opsi perintah Iperf dengan ukuran *window* lebih kecil dari standar yaitu 42 KB.

2. **Tahapan kedua** :

   Pengujian terhadap parameter *jitter* dan *packet loss* menggunakan parameter UDP. Pengujian ini juga menggunakan opsi perintah Iperf sesuai dengan konfigurasi Iperf pada pengujian. Tahap ini memiliki dua skenario dan

masing masing dilakukan sebanyak lima kali pengujian. Dan dua skenario tersebut adalah :

a. **Skenario 1** : Pengujian *jitter* dan *packet loss* dengan menggunakan parameter batas *bandwidth* 100 Mbps

b. **Skenario 2** : Pengujian *jitter* dan *packet loss* dengan menggunakan parameter batas *bandwidth* 1024 Mbps

3. **Tahapan ketiga** :

Pengujian tahap ini adalah pengujian terhadap pemakaian CPU. Pengujian ini melakukan perekaman aktivitas CPU yang dilakukan pada tahap pertama. Artinya pengujian ini tergantung pada pengujian pertama dan memiliki jumlah pengujian yang sama yaitu lima kali. Pengujian ini melakukan perekaman CPU ketika kontroler sedang dinyalakan. Pengujian dilakukan selama satu menit dengan sepuluh detik pertama kontroler dalam keadaan *idle* dilanjutkan empat puluh detik selanjutnya pengujian parameter QoS dan sepuluh detik terakhir keadaan *idle* setelah pengujian parameter QoS.

Ringkasan dari seluruh perancangan skenario pengujian dapat dilihat seperti pada Tabel 3.8 sebagai berikut :

Tabel 3. 8 Ringkasan perancangan skenario pengujian

Tahapan	Paramater Uji	Kontroler	Protokol	Jumlah Skenario	Opsi Iperf	Jumlah Pengujian	Total Pengujian
1	*Throughput* dan *Bandwidth*	*Ryu* dan *pyretic*	TCP	3	-w	5	2 x 3 x 5
2	*Jitter* dan *Packet loss*	*Ryu* dan *pyretic*	UDP	2	-b	5	2 x 2 x 5
3	Pemakaian CPU dan RAM	*Ryu* dan *pyretic*	Pengujian ini dilakukan selama 60 detik. 10 detik pertama pada keadaan *idle,* 40 detik selanjutnya keadaan uji QoS dan 10 detik terakhir keadaan *idle* setelah pengujian QoS.				2 x 5

Pengujian yang dilakukan akan menghasilkan banyak data yang bervariasi. Untuk mendapatkan data tunggal, maka setiap data dari pengujian dikalkulasi untuk menentukan data rata-rata. Artinya hasil pengujian yang didapatkan sebanyak lima akan dirata-rata menjadi data tunggal.

BAB 4 HASIL

Bab ini menjelaskan implementasi lingkungan pengujian, pengujian dan hasil dari seluruh pengujian dua kontroler *Ryu* dan *Pyretic* yang telah dilakukan sesuai dengan skenario pada bab sebelumnya. Implementai Lingkungan mencakup instalasi perangkat lunak pengujian dan kode sumber algoritma *Non-weighted Dijkstra*. Pada penelitian ini pengujian dilakukan menggunakan tiga tahap dan beberapa skenario sesuai dengan skenario pengujian pada perancangan. Tahap pertama pengujian terhadap *throughput* dan *bandwidth*, tahap kedua pengujian terhadap *jitter* dan *packet loss*, dan yang ketiga pengamatan pemakaian CPU terhadap pengujian pertama. Seluruh pengujian melibatkan dua kontroler yaitu *Ryu* dan *Pyretic* yang menerapkan algoritma *Non-weighted Dijkstra.* Dan hasil pengujian menampilkan hasil uji seluruh parameter yang diuji pada setiap kontroler. Hasil pengujian ditampilkan berdasarkan dengan urutan skenario sesuai dengan perancangan.

4.1 Implementasi Lingkungan Pengujian

Implementasi dilakukan sesuai dengan rancangan yang telah dibuat pada bab sebelumnya. Implementasi lingkungan pengujian merupakan penerapan terhadap seluruh lingkungan pengujian yang dibutuhkan sesuai dengan rancangan. Beberapa hal yang diperlukan untuk menerapkan sistem adalah mempersiapkan perangkat keras dan instalasi perangkat lunak.

4.1.1 Instalasi Mininet

Mininet adalah sebuah emulator jaringan *open source* yang mendukung protokol *OpenFlow* untuk arsitektur SDN. Berikut adalah langkah instalasi Mininet:

1. Unduh berkas Mininet pada website github dengan perintah

```
$ git clone git://github.com/download/Mininet/Mininet
```

2. Setelah berkas selesai diunduh maka masuk ke direktori Mininet dengan perintah :

```
$ cd Mininet
```

kemudian periksa versi Mininet yang akan diinstal dengan perintah :

```
$ git tag
```

3. Untuk mengaktifkan versi Mininet yang akan diinstal dapat dilakukan dengan perintah

```
$ git checkout <versi Mininet>
```

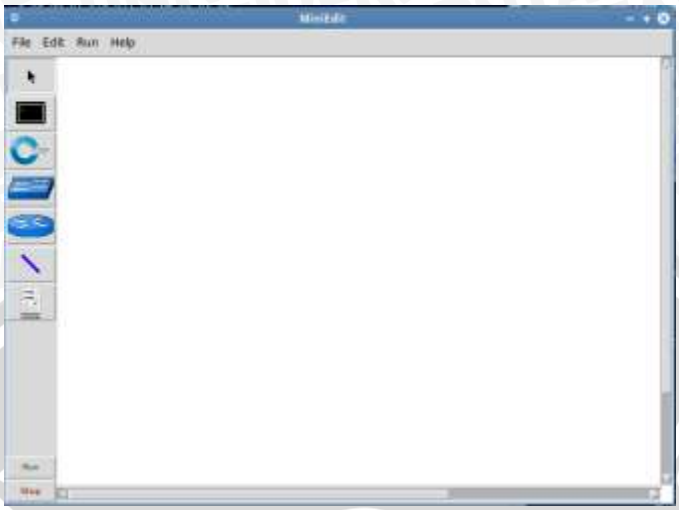
4. Langkah selanjutnya adalah mengeksekusi berkas install.sh dengan perintah

```
$ ./Mininet/util/install.sh -a
```

5. Setelah semua langkah-langkah instalasi dilakukan, maka emulator *Mininet* dapat dijalankan melalui GUI dengan menjalankan berkas *miniedit.py* yang berada pada direktori */Mininet/example*. Dengan cara :

```
$ cd Mininet/example
  sudo ./miniedit.py
```

Lalu miniedit akan muncul seperti pada gambar 4.1



Gambar 4. 1 Mininet dalam bentuk GUI atau miniedit

4.1.2 Instalasi Kontroler *Ryu*

Untuk melakukan instalasi kontroler *Ryu* pada sistem, dapat dilakukan dengan mengikuti penjelasan berikut ini:

1. Sebelum menginstal *Ryu*, pertama pasang dulu paket python library

```
$ sudo apt-get install python-pip
$ sudo apt-get install python-dev
```

2. Lalu pasang paket dependensi *Ryu*

```
$ sudo apt-get install python-eventlet
$ sudo apt-get install python-routes
$ sudo apt-get install python-webob
$ sudo apt-get install python-paramiko
```

3. Gunakan pip, lalu instal *Ryu*

```
$ sudo pip install Ryu
```

4. Lakukan pembaharuan pip six suites, karena versi lama akan membuat *Ryu* jadi error.

```
$ sudo pip install six –upgrade
```

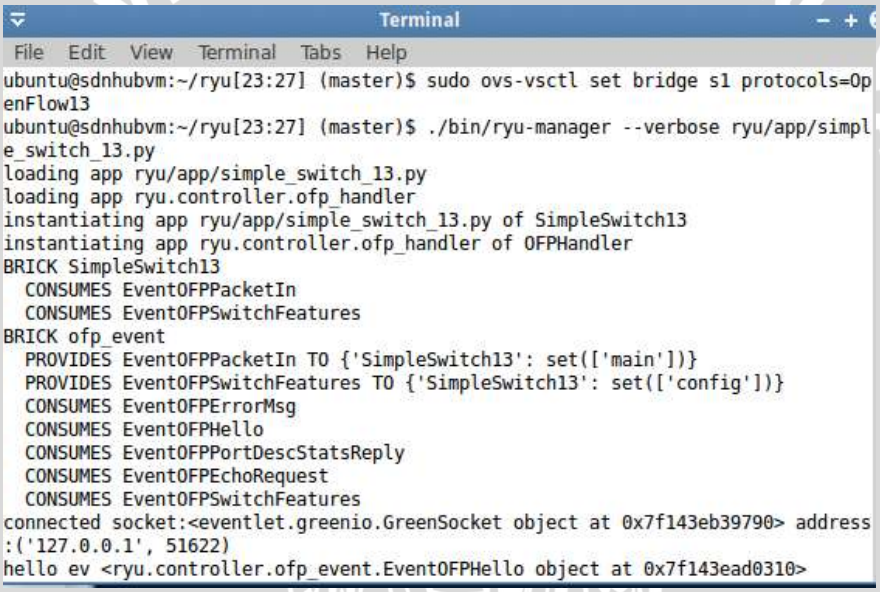
5. setelah *Pyretic* terpasang, selanjutnya pastikan bahwa kontroler dapat dijalankan. Untuk menjalankan *Ryu*, jalankan topologi standar pada Mininet menggunakan terminal. Contoh topologi yang dapat digunakan adalah dengan perintah berikut:

```
$ sudo mn --topo single, 3 --mac --controller remote --switch ovsk
```

6. Selanjutnya jalankan kontroler *Ryu* dengan cara mengeksekusi perintah berikut ini pada terminal. Buka terminal dan pastikan Anda berada pada direktori *Ryu*, lalu jalankan

```
$  sudo ovs-vsctl set bridge s1 protocols=OpenFlow13
$ ./bin/Ryu-manager --verbose Ryu/app/simple_switch_13.py
```

7. Pada perintah tersebut *Ryu* menjalankan aplikasi yang sudah ada pada modul secara default yaitu *simple_switch_13.py*. Berikut adalah kontroler *Ryu* yang sukses dijalankan seperti pada Gambar 4.2:



Gambar 4. 2 Kontroler *Ryu* yang telah berjalan

4.1.3 Instalasi Kontroler *Pyretic*

Untuk dapat menggunakan kontroler *Pyretic* pada sistem, dibutuhkan instalasi terlebih dahulu. Berikut ini adalah langkah instalasi kontroler *Pyretic* pada Xubuntu:

1. Pasang dependesi python yang dibutuhkan

```
$ sudo apt-get install python-dev python-pip python-netaddr screen hping3 ml-lpt graphviz ruby1.9.1-dev libboost-dev libboost-test-dev libboost-program-options-dev libevent-dev automake libtool flex bison pkg-config g++ libssl-dev python-all python-all-dev python-all-dbg
```

```
$ sudo pip install networkx bitarray netaddr ipaddr pytest
  ipdb sphinx pyparsing==1.5.7 yappi
$ sudo gem install jekyll
```

2. Berikan patch dependensi *asynchat python*

```
$ wget https://raw.github.com/frenetic-
  lang/Pyretic/master/Pyretic/backend/patch/asynchat.py
$ sudo mv asynchat.py /usr/lib/python2.7/
$ sudo chown root:root /usr/lib/python2.7/asynchat.py
```

3. Pasang git *subtree*

```
$ git clone https://github.com/git/git.git
$ pushd git/contrib/subtree/
$ make
$ mv git-subtree.sh git-subtree
$ sudo install -m 755 git-subtree /usr/lib/git-core
$ popd
$ rm -rf git
```

4. Duplikasi repositori *Pyretic* ke direktori *home*

```
$ cd ~
$ git clone http://github.com/frenetic-lang/Pyretic.git
```

5. Atur variabel *environtment* dengan menambahkan baris berikut di akhir dari *.profile*

```
$ export PATH=$PATH:$HOME/Pyretic:$HOME/pox
$ export PYTHONPATH=$HOME/Pyretic:$HOME/Mininet:$HOME/pox
```

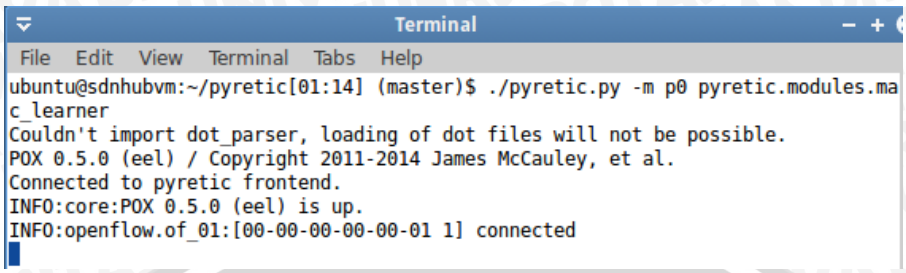
6. Setelah *Pyretic* terpasang, selanjutnya pastikan bahwa kontroler dapat dijalankan. Untuk menjalankan *Pyretic* , jalankan topologi standar pada Mininet menggunakan terminal. Contoh topologi yang dapat digunakan adalah dengan perintah berikut:

```
$ sudo mn --topo single, 3 --mac --controller remote --switch
  ovsk
```

7. Untuk menjalankan *Pyretic*, jalankan perintah berikut ini pada terminal. Buka terminal dan pastikan Anda berada pada direktori *Pyretic*, lalu jalankan

```
$ ./Pyretic.py -m p0 Pyretic.modules.mac_learner.
```

8. Pada perintah tersebut *Pyretic* menjalankan aplikasi yang sudah ada pada modul yaitu *mac_learner*. Berikut adalah bahwa *Pyretic* sukses dijalankan seperti pada Gambar 4.3 :



Gambar 4. 3 Kontroler *Pyretic* yang telah dijalankan

4.1.4 Instalasi Iperf

Instalasi Ipef sangat mudah, terlebih jika menggunakan sistem operasi linux khususnya ubuntu dan turunannya. Iperf dipasang pada sistem operasi yang digunakan dalam pengujian. Berikut ini adalah langkah memasang iperf di Xubuntu:

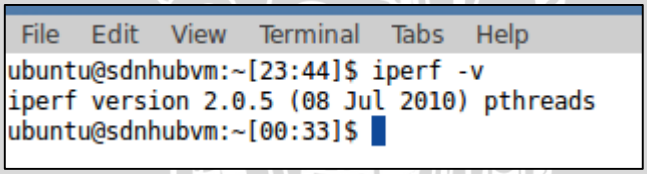
1. Ketikkan pada terminal dan tunggu hingga instalasi selesai

```
$ sudo apt-get install iperf
```

2. Setelah instalasi selesai cek apakah iperf telah terpasang dengan baik. Jalankan perintah berikut

```
$ Iperf -v
```

Terminal akan menunjukkan tampilan seperti Gambar 4.4 berikut :



Gambar 4. 4 Versi Iperf yang digunakan untuk pengujian

4.1.5 Algoritma *Non-weighted Dijkstra*

Untuk menjalankan aplikasi dijkstra pada kontroler dibutuhkan *source code* untuk dijalankan, sayangnya secara *default* kedua kontroler tidak menyediakan pada aplikasi tersebut. *Non-weighted Dijkstra* adalah algoritma yang digunakan untuk menjari jalur tercepat menuju tujuan. Dan implementasi *Non-weighted Dijkstra* pada pengujian ini adalah mencari path terpendek dengan menghitung jumlah *hop* (*switch*) yang ada pada topologi. Maka perlu diberikan penyesuaian *kode sumber* pada direktori aplikasi pada masing masing kontroler.

4.1.5.1 *Non-Weighted Dijkstra* pada *Ryu*

Agar dapat menjalankan aplikasi dijkstra pada *Ryu* dibutuhkan penyesuaian dengan lingkungan pengujian. kode sumber Dr. Chih-Heng Ke. Berikut algoritma Non-Weighted Dijkstra pada *Ryu* :

Tabel 4.1 Kode sumber *Non-weighted Dijkstra Ryu*

```
57    for dpid in switches:
58      distance[dpid] = float('Inf')
59      previous[dpid] = None
61    distance[src]=0
62    Q=set(switches)
63    print "Q=", Q
65    while len(Q)>0:
66      u = minimum_distance(distance, Q)
67      Q.remove(u)
69      for p in switches:
70        if adjacency[u][p]!=None:
71          w = 1
72          if distance[u] + w < distance[p]:
73            distance[p] = distance[u] + w
74            previous[p] = u
```

Implementasi algoritma *non-weighted dijkstra* dilakukan dengan menggunakan bahasa *python*. Implementasi algoritma tersebut diterapkan pada kedua kontroler. Implementasi pada kontroler Ryu dapat dilihat pada tabel 4.1. Penjelasan algoritma sesuai dengan gambar 4.1 adalah sebagai berikut:

Baris 57 : insialisasi dpid pada seluruh data switch

Baris 58 : mendefinisikan nilai *infinity distance* dari node awal ke tujuan

Baris 59 : mendefinisikan node awal dari source

Baris 61-63 : nilai *distance* src=0, definisikan Q sebagai kumpulan data *switch* dan cetak Q

Baris 66 : inisialisasi nilai u dari minimum_distance dan node dipilih duluan

Baris 67 : menhilagkan nilai u pada Q

Baris 70-74 : pemilihan nilai terkecil yang dibandingkan pada baris 72 dengan nilai w=1 (bobot)

Implementasi *non-weighted dijkstra* membutuhkan dependensi *library* yang tepat. Penggunaan *library* pada kode memberikan fitur dari kontroler tanpa membuat kode dari awal. Library yang digunakan dalam kode sumber pada kontroler *Ryu* dapat dilihat pada tabel 4.2.

Tabel 4. 2 Library pada kode sumber *non-weighted dijkstra* kontroler *Ryu*

```
16  from ryu.base import app_manager
17  from ryu.controller import mac_to_port
18  from ryu.controller import ofp_event
19  from ryu.controller.handler import CONFIG_DISPATCHER,
    MAIN_DISPATCHER
20  from ryu.controller.handler import set_ev_cls
21  from ryu.ofproto import ofproto_v1_3
22  from ryu.lib.mac import haddr_to_bin
23  from ryu.lib.packet import packet
24  from ryu.lib.packet import ethernet
25  from ryu.lib.packet import ether_types
26  from ryu.lib import mac
28  from ryu.topology.api import get_switch, get_link
29  from ryu.app.wsgi import ControllerBase
30  from ryu.topology import event, switches
31  from collections import defaultdict
```

4.1.5.2 *Non-Weighted Dijkstra* pada *Pyretic*

Agar dapat menjalankan aplikasi dijkstra pada *Pyretic* dibutuhkan penyesuaian dengan lingkungan pengujian. Kode sumber tersebut dibuat oleh Dr. Chih-Heng Ke. Dan berikut adalah kode sumber *Non-Weighted Dijkstra Pyretic* seperti pada tabel 4.3.

Tabel 4. 3 Kode sumber *Non-weighted Dijkstra Pyretic*

```
31      For dpid in switches:
32        distance[dpid] = 9999
33        previous[dpid] = None
35      distance[src]=0
36      Q=set(switches)
38      while len(Q)>0:
39        u = minimum_distance(distance, Q)
40        Q.remove(u)
42        for p in switches:
43          if adjacency[u][p]!=None:
44            w = 1
45            if distance[u] + w < distance[p]:
46              distance[p] = distance[u] + w
47              previous[p] = u
```

Implementasi algoritma *non-weighted dijkstra* dilakukan dengan menggunakan bahasa *python*. Implementasi algoritma tersebut diterapkan pada kedua kontroler. Implementasi pada kontroler *Pyretic* dapat dilihat pada tabel 4.1. Penjelasan algoritma sesuai dengan gambar 4.2 adalah sebagai berikut:

Baris 31 : insialisasi dpid pada seluruh data switch

Baris 32 : mendefinisikan nilai *infinity distance* dari node awal ke tujuan

Baris 33 : mendefinisikan node awal dari source

Baris 35-36 : nilai *distance* src=0, definisikan Q sebagai kumpulan data *switch*

Baris 39 : inisialisasi nilai u dari minimum_distance dan node dipilih duluan

Baris 40 : menhilagkan nilai u pada Q

Baris 42-47 : pemilihan nilai terkecil yang dibandingkan pada baris 72 dengan nilai w=1 (bobot)

Implementasi *non-weighted dijkstra* membutuhkan dependensi *library* yang tepat. Penggunaan *library* pada kode memberikan fitur dari kontroler tanpa membuat kode dari awal. Library yang digunakan dalam kode sumber pada kontroler *Ryu* dapat dilihat pada tabel 4.4

Tabel 4. 4 Library pada kode sumber *non-weighted dijkstra* kontroler Pyretic

```
1  from pyretic.lib.corelib import*
2  from pyretic.lib.std import *
3  from multiprocessing import Lock
4  from pyretic.lib.query import *
5  from collections import defaultdict
```

.

4.2 Hasil Pengujian Tahap Pertama Skenario 1

Pengujian tahap pertama skenario 1 adalah pengujian terhadap performa *throughput* dan *bandwidth* pada kontroler *Ryu* dan *Pyretic*. Pengujian performa *throughput* dan *bandwidth* menggunakan 9 *client* terhadap 1 *server* dan masing-masing diuji sebanyak lima kali secara sekuensial ditiap kontroler. Kontroler pertama yang diuji adalah *Ryu* dan dilanjutkan pengujian terhadap kontroler *Pyretic* setelahnya. Penjelasan ringkas pada pengujian tahap ini dapat dilihat seperti pada tabel 4.5.

Tabel 4. 5 Rangkuman pengujian tahap pertama skenario 1

Paramater Uji	Protokol	*TCP Window Size*	Kontroler	Aplikasi Kontroler	Waktu Pengujian	Jumlah Pengujian
Throughput dan *Bandwidth*	TCP	Standar : 85,3 KB	*Ryu* dan *Pyretic*	*Non-weighted Dijkstra*	10 detik tiap parameter	5 kali per kontroler

Pengujian tahap pertama :

Skenario 1 : Pengujian *throughput* dan *bandwidth* dengan opsi perintah Iperf dengan ukuran *window* standar yaitu 85,3 KB. Perintah pada iperf seperti pada tabel 4.6 :

Tabel 4. 6 Perintah iperf pada pengujian tahap pertama skenario 1

Server	*Client*
Iperf –s –u –i 1	Iperf –c 10.0.0.1 –f m

Pengujian performa *throughput* dan *bandwidth* menggunakan aplikasi Iperf. Untuk pengujian dilakukan dengan perintah iperf seperti pada tabel 4.4 dan salah satu hasilnya dapat dilihat seperti pada gambar 4.5 berikut :

```
root@sdnhubvm:~/mininet/examples[23:07] (master)$ iperf -c 10.0.0.1 -f m
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 0.08 MByte (default)
------------------------------------------------------------
[  5] local 10.0.0.2 port 56599 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  9469 MBytes  7943 Mbits/sec
```

Gambar 4. 5 Hasil pengujian *throughput* dan *bandwidth* percobaan 1 pada kontroler *Ryu* dari *host* 2 ke *server* skenario 1

hasil dari pengujian *throughput* diperoleh dengan mengkalkulasi jumlah *Byte* yang diterima dibagi dengan waktu transmisi.

4.2.1 *Throughput* dan *Bandwidth* Pada *Ryu* skenario 1

Pada sub bab ini hasil pengujian performa *throughput* dan *bandwidth* terhadap kontroler *Ryu* menggunakan Iperf dengan ukuran *TCP window* sebesar 85,3 KB. Pengujian dilakukan sebanyak lima kali secara sekuensial. Dan hasil pengujian dirangkum seperti pada tabel 4.7

Tabel 4. 7 Hasil pengukuran *throughput* dan *bandwidth Ryu* skenario 1

Host	Tujuan	Waktu	*Throughput* (MBps)	*Bandwidth* (Mbps)	Jalur / Path
2	1	10s	553,9	4645,8	2-8-7-3-1
3	1	10s	757,2	6351,4	3-1
4	1	10s	708,6	5943,6	4-6-1
5	1	10s	578,6	4853,6	5-7-3-1
6	1	10s	739,8	6204,8	6-1
7	1	10s	657,3	5513,0	7-3-1
8	1	10s	562,8	4620,0	8-7-3-1
9	1	10s	520,3	4364,0	9-5-7-3-1
10	1	10s	477,3	4004,0	10-9-5-7-3-1

Setelah pengujian dilakukan, hasil dari lima kali pengujian dikalkulasikan menjadi data rata-rata seperti pada tabel 4.7. Pembahasan terhadap hasil pengujian akan dijelaskan pada bab selanjutnya.

4.2.2 *Throughput* dan *Bandwidth* Pada *Pyretic* skenario 1

Pada sub bab ini hasil pengujian performa *throughput* dan *bandwidth* terhadap kontroler *Pyretic* menggunakan Iperf dengan ukuran *TCP window* sebesar 85,3 KB. Pengujian dilakukan sebanyak lima kali secara sekuensial. Dan hasil pengujian dirangkum seperti pada tabel 4.8 berikut :

Tabel 4. 8 Hasil pengukuran *throughput* dan *bandwidth Pyretic* skenario 1

Host	Tujuan	Waktu (s)	*Throughput (MBps)*	*Bandwidth* (Mbps)	Jalur / Path
2	1	10s	905,8	7598,4	2-8-7-3-1
3	1	10s	1125,1	9438,0	3-1
4	1	10s	1067,8	8957,2	4-6-1
5	1	10s	1088,6	9131,8	5-7-3-1
6	1	10s	1096,9	9208,2	6-1
7	1	10s	1026,1	8607,4	7-3-1
8	1	10s	950,3	7971,8	8-7-3-1
9	1	10s	864,5	7251,4	9-5-7-3-1
10	1	10s	832,6	6985,0	10-9-5-7-3-1

setelah pengujian dilakukan, hasil dari lima kali pengujian dikalkulasikan menjadi data rata-rata seperti pada tabel 4.8. Pembahasan terhadap hasil pengujian akan dijelaskan pada bab selanjutnya.

4.3 Hasil Pengujian Tahap Pertama Skenario 2

Pengujian tahap pertama skenario 2 adalah pengujian terhadap performa *throughput* dan *bandwidth* pada kontroler *Ryu* dan *Pyretic* dengan menggunakan ukuran *TCP window* sebesar 170 KB selama 10 detik. Pengujian performa *throughput* dan *bandwidth* menggunakan 10 *client* terhadap 1 *server* dan masing-masing diuji sebanyak lima kali secara sekuensial pada tiap kontroler. Kontroler pertama yang diuji adalah *Ryu* dan dilanjutkan pengujian terhadap kontroler *Pyretic* setelahnya. Penjelasan ringkas pada pengujian tahap ini dapat dilihat seperti pada tabel 4.9 berikut:

Tabel 4. 9 Rangkuman pengujian tahap pertama skenario 2

Paramater Uji	Protokol	*TCP Window Size*	Kontroler	Aplikasi Kontroler	Waktu Pengujian	Jumlah Pengujian
Throughput dan *Bandwidth*	TCP	170 KB	*Ryu* dan *Pyretic*	*Non-weighted Dijkstra*	10 detik tiap parameter	5 kali per kontroler

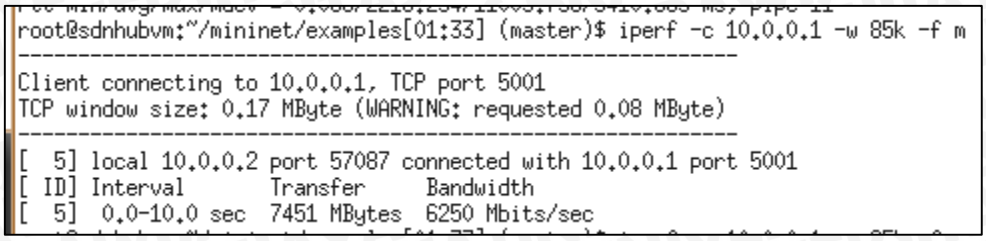
Pengujian tahap pertama skenario 2:

Skenario 2 : Pengujian *throughput* dan *bandwidth* dengan opsi perintah Iperf dengan ukuran *window* lebih besar dari standar yaitu 170 KB. Perintah pada Iperf seperti pada tabel 4.10 :

Tabel 4. 10 Perintah iperf pada pengujian tahap pertama skenario 2

Server	*Client*
Iperf –s –u	Iperf –c 10.0.0.1 –w 85k –f m

Pengujian performa *throughput* dan *bandwidth* dilakukan dengan menggunakan *TCP window* sebesar 170 KB. Namun agar ukuran sesuai dengan yang diinginkan, maka perintah pada iperf diganti dengan nilai 85 KB karena nilai tersebut akan dikali dua oleh Iperf. Untuk pengujian dilakukan dengan perintah Iperf seperti pada tabel 4.10 dan salah satu hasilnya dapat dilihat seperti pada gambar 4.6.



Gambar 4. 6 Hasil pengujian *throughput* dan *bandwidth* kontroler *Ryu host* 2 ke *server* skenario 2

4.3.1 *Throughput* dan *Bandwidth* Pada *Ryu* skenario 2

Pada sub bab ini hasil pengujian performa *throughput* dan *bandwidth* terhadap kontroler *Ryu* menggunakan Iperf dengan ukuran *TCP window* sebesar 170 KB KB. Pengujian dilakukan sebanyak lima kali secara sekuensial. Dan hasil pengujian dirangkum seperti pada tabel 4.11 berikut :

Tabel 4. 11 Hasil pengukuran *throughput* dan *bandwidth Ryu* skenario 2

Host	Tujuan	Waktu (s)	*Throughput (MBps)*	*Bandwidth* (Mbps)	Jalur / Path
2	1	10s	439,8	3243,2	2-8-7-3-1
3	1	10s	596,6	5919,6	3-1
4	1	10s	426,2	4033,0	4-6-1
5	1	10s	483,6	4675,2	5-7-3-1
6	1	10s	664,5	5574,0	6-1
7	1	10s	624,7	5240,2	7-3-1
8	1	10s	561,6	4710,6	8-7-3-1
9	1	10s	499,8	4192,8	9-5-7-3-1
10	1	10s	362,3	3039,6	10-9-5-7-3-1

setelah pengujian dilakukan, hasil dari lima kali pengujian dikalkulasikan menjadi data rata-rata seperti pada tabel 4.11. Pembahasan terhadap hasil pengujian akan dijelaskan pada bab selanjutnya.

4.3.2 *Throughput* dan *Bandwidth* Pada *Pyretic* skenario 2

Pada sub bab ini hasil pengujian performa *throughput* dan *bandwidth* terhadap kontroler *Pyretic* menggunakan Iperf dengan ukuran *TCP window* sebesar 170 KB. Pengujian dilakukan sebanyak lima kali secara sekuensial. Dan hasil pengujian dirangkum seperti pada tabel 4.12.

Tabel 4. 12 Hasil pengukuran *throughput* dan *bandwidth Pyretic* skenario 2

Host	Tujuan	Waktu (s)	*Throughput (MBps)*	*Bandwidth* (MBps)	Jalur / Path
2	1	10s	526,0	4412,0	2-8-7-3-1
3	1	10s	679,5	5996,6	3-1
4	1	10s	606,3	5350,0	4-6-1
5	1	10s	607,4	6081,6	5-7-3-1
6	1	10s	773,5	6488,2	6-1
7	1	10s	682,3	5724,0	7-3-1
8	1	10s	665,6	5583,0	8-7-3-1
9	1	10s	575,9	4830,8	9-5-7-3-1
10	1	10s	434,3	3642,8	10-9-5-7-3-1

4.4 Hasil Pengujian Tahap Pertama Skenario 3

Pengujian tahap pertama skenario 3 adalah pengujian terhadap performa *throughput* dan *bandwidth* pada kontroler *Ryu* dan *Pyretic* dengan menggunakan ukuran *TCP window* sebesar 42 KB selama 10 detik. Pengujian performa *throughput* dan *bandwidth* menggunakan 9 *client* terhadap 1 *server* dan masing-masing diuji sebanyak lima kali secara sekuensial ditiap kontroler. Kontroler pertama yang diuji adalah *Ryu* dan dilanjutkan pengujian terhadap kontroler *Pyretic* setelahnya. Penjelasan ringkas pada pengujian tahap ini dapat dilihat seperti pada tabel 4.13 berikut:

Tabel 4. 13 Rangkuman pengujian tahap pertama skenario 3

Paramater Uji	Protokol	*TCP Window Size*	Kontroler	Aplikasi Kontroler	Waktu Pengujian	Jumlah Pengujian
Throughput dan *Bandwidth*	TCP	42 KB	*Ryu* dan *Pyretic*	*Non-weighted Dijkstra*	10 detik tiap parameter	5 kali per kontroler

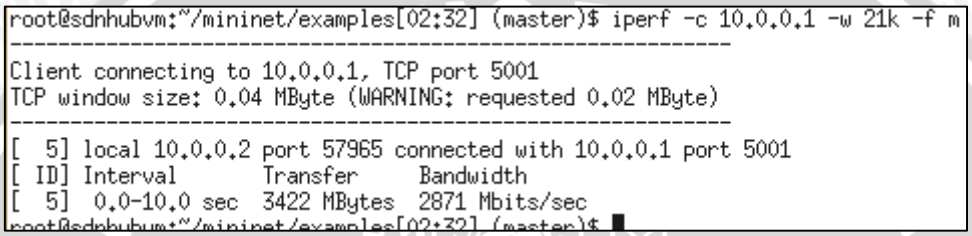
Pengujian tahap pertama skenario 3 :

Skenario 3 : Pengujian *throughput* dan *bandwidth* dengan opsi perintah Iperf dengan ukuran *window* lebih kecil dari standar yaitu 42 KB. Perintah pada iperf seperti pada tabel 4.14.

Tabel 4. 14 Perintah iperf pada pengujian tahap pertama skenario 3

Server	*Client*
Iperf –s –u –i 1	Iperf –c 10.0.0.1 –w 21k –f m

Pengujian performa *throughput* dan *bandwidth* dilakukan dengan menggunakan *TCP window* sebesar 42 KB. Namun agar ukuran sesuai dengan yang diinginkan, maka perintah pada iperf diganti dengan nilai 21 KB karena nilai tersebut akan dikali dua oleh Iperf. Untuk pengujian dilakukan dengan perintah Iperf seperti pada tabel 4.14 dan salah satu hasilnya dapat dilihat seperti pada gambar 4.7 berikut :



Gambar 4. 7 Hasil pengujian *throughput* dan *bandwidth* kontroler *Ryu host* 2 ke *server* skenario 2

4.4.1 *Throughput* dan *Bandwidth* Pada *Ryu* skenario 3

Pada sub bab ini hasil pengujian performa *throughput* dan *bandwidth* terhadap kontroler *Ryu* menggunakan Iperf dengan ukuran *TCP window* sebesar 42 KB. Pengujian dilakukan sebanyak lima kali secara sekuensial. Dan hasil pengujian dirangkum seperti pada tabel 4.15 berikut :

Tabel 4. 15 Hasil pengukuran *throughput* dan *bandwidth Ryu* skenario 3

Host	Tujuan	Waktu (s)	*Throughput (MBps)*	*Bandwidth* (MBps)	Jalur / Path
2	1	10s	222,4	1897,6	2-8-7-3-1
3	1	10s	260,8	2187,2	3-1
4	1	10s	234,5	1967,2	4-6-1
5	1	10s	221,3	1856,8	5-7-3-1
6	1	10s	261,9	2196,6	6-1
7	1	10s	240,3	2016,2	7-3-1
8	1	10s	231,7	1943,2	8-7-3-1
9	1	10s	207,5	1740,4	9-5-7-3-1
10	1	10s	189,7	1591,6	10-9-5-7-3-1

setelah pengujian dilakukan, hasil dari lima kali pengujian dikalkulasikan menjadi data rata-rata seperti pada tabel 4.13. Pembahasan terhadap hasil pengujian akan dijelaskan pada bab selanjutnya.

4.4.2 *Throughput* dan *Bandwidth* Pada *Pyretic* skenario 3

Pada sub bab ini hasil pengujian performa *throughput* dan *bandwidth* terhadap kontroler *Pyretic* menggunakan Iperf dengan ukuran *TCP window* sebesar 42 KB. Pengujian dilakukan sebanyak lima kali secara sekuensial. Dan hasil pengujian dirangkum seperti pada tabel 4.16.

Tabel 4. 16 Hasil pengukuran *throughput* dan *bandwidth Pyretic* skenario 3

Host	Tujuan	Waktu (s)	*Throughput (MBps)*	*Bandwidth* (MBps)	Jalur / Path
2	1	10s	236,4	1983,4	2-8-7-3-1
3	1	10s	274,4	2301,8	3-1
4	1	10s	258,4	2167,6	4-6-1
5	1	10s	250,6	2102,4	5-7-3-1
6	1	10s	274,3	2300,8	6-1
7	1	10s	250,1	2162,4	7-3-1
8	1	10s	235,6	1976,2	8-7-3-1
9	1	10s	223,7	1876,0	9-5-7-3-1
10	1	10s	223,3	1873,0	10-9-5-7-3-1

4.5 Hasil Pengujian Tahap Kedua Skenario 1

Pengujian tahap kedua skenario 1 adalah pengujian terhadap performa *jitter* dan *packet loss* pada kontroler *Ryu* dan *Pyretic*. Pengujian performa *jitter* dan *packet loss* menggunakan 9 *client* terhadap 1 *server* dan masing-masing diuji sebanyak lima kali secara sekuensial ditiap kontroler. Kontroler pertama yang diuji adalah *Ryu* dan dilanjutkan pengujian terhadap kontroler *Pyretic* setelahnya. Penggunaan batasan *bandwidth* dilakukan dengan berdasarkan dokumentasi pengujian Iperf. Batasan *bandwidth* digunakan untuk skenario 1 adalah 100 Mbps. Penjelasan ringkas pada pengujian tahap ini dapat dilihat seperti pada tabel 4.17

.

Tabel 4. 17 Rangkuman pengujian tahap kedua skenario 1

Paramater Uji	Protokol	*Bandwidth Size*	Kontroler	Aplikasi Kontroler	Waktu Pengujian	Jumlah Pengujian
Jitter dan *Packet loss*	UDP	100 Mbps	*Ryu* dan *Pyretic*	*Non-weighted Dijkstra*	10 detik tiap parameter	5 kali per kontroler

Pengujian tahap kedua skenario 1 :

Skenario 1 : Pengujian *jitter* dan *packet loss* dengan menggunakan parameter batas *bandwidth* 100 Mbps. Perintah pada iperf seperti pada tabel 4.18.

Tabel 4. 18 Perintah iperf pada pengujian tahap kedua skenario 1

Server	*Client*
Iperf –s –u	Iperf –c 10.0.0.1 –b 100m

Pengujian performa *jitter* dan *packet loss* dilakukan dengan menggunakan *batas bandwidth* sebesar 100 Mbps. Untuk pengujian dilakukan dengan perintah Iperf seperti pada tabel 4.18 dan salah satu hasilnya dapat dilihat seperti pada gambar 4.8 berikut :

```
root@sdnhubvm:~/mininet/examples[06:43] (master)$ iperf -c 10.0.0.1 -b 100m
WARNING: option -b implies udp testing
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.4 port 36610 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec   119 MBytes   100 Mbits/sec
[  5] Sent 85112 datagrams
[  5] Server Report:
[  5]  0.0-10.0 sec   119 MBytes   100 Mbits/sec   0.014 ms   69/85111 (0.081%)
[  5]  0.0-10.0 sec  1 datagrams received out-of-order
```

Gambar 4. 8 Hasil pengujian *throughput* dan *bandwidth* kontroler *Ryu host* 2 ke *server* skenario 2

4.5.1 *Jitter* dan *Packet loss* Pada *Ryu* skenario 1

Pada tahap ini pengujian dilakukan terhadap performa *jitter* dan *packet loss* pada kontroler *Ryu* dengan menggunakan waktu selama 10 detik dan pengujian menggunakan protokol UDP. Iperf digunakan dengan membuat satu *server* dan 9 *client* untuk uji *jitter.* Pengujian menggunakan nilai *bandwidth* sebesar 100 Mbps. Pengujian dilakukan sebanyak lima kali secara sekuensial. Dan hasil pengujian dirangkum seperti pada tabel 4.19.

Tabel 4. 19 Hasil pengukuran *jitter* dan *packet loss Ryu* pada batas *bandwidth* 100Mbps

Host	Tujuan	Waktu	*Jitter* (ms)	*Packet loss* (%)	Jalur / Path
2	1	10s	0,019	0,055	2-8-7-3-1
3	1	10s	0,013	0,030	3-1
4	1	10s	0,022	0,041	4-6-1
5	1	10s	0,012	0,049	5-7-3-1
6	1	10s	0,030	0,028	6-1
7	1	10s	0,013	0,047	7-3-1
8	1	10s	0,027	0,051	8-7-3-1
9	1	10s	0,023	0,062	9-5-7-3-1
10	1	10s	0,016	0,075	10-9-5-7-3-1

setelah pengujian dilakukan, hasil dari lima kali pengujian dikalkulasikan menjadi data rata-rata seperti pada tabel 4.19. Pembahasan terhadap hasil pengujian akan dijelaskan pada bab selanjutnya.

4.5.2 *Jitter* dan *Packet loss* Pada *Pyretic* skenario 1

Pada tahap ini pengujian dilakukan terhadap performa *jitter* dan *packet loss* pada kontroler *Pyretic* dengan menggunakan waktu selama 10 detik dan pengujian menggunakan protokol UDP. Iperf digunakan dengan membuat satu *server* dan 9 *client* untuk uji *jitter.* Pengujian menggunakan nilai *bandwidth* sebesar 100 Mbps. Pengujian dilakukan sebanyak lima kali secara sekuensial. Dan hasil pengujian dirangkum seperti pada tabel 4.20 berikut :

Tabel 4. 20 Hasil pengukuran *jitter Pyretic* pada nilai *bandwidth* 100Mbps

Host	Tujuan	Waktu	*Jitter* (ms)	*Packet loss* (%)	Jalur / Path
2	1	10s	0,018	0,041	2-8-7-3-1
3	1	10s	0,014	0,015	3-1
4	1	10s	0,012	0,023	4-6-1
5	1	10s	0,010	0,031	5-7-3-1
6	1	10s	0,013	0,019	6-1
7	1	10s	0,012	0,028	7-3-1
8	1	10s	0,017	0,032	8-7-3-1

Host	Tujuan	Waktu	*Jitter* (ms)	*Packet loss* (%)	Jalur / Path
9	1	10s	0,009	0,036	9-5-7-3-1
10	1	10s	0,013	0,044	10-9-5-7-3-1

Setelah pengujian dilakukan, hasil dari lima kali pengujian dikalkulasikan menjadi data rata-rata seperti pada tabel 4.20. Pembahasan terhadap hasil pengujian akan dijelaskan pada bab selanjutnya.

4.6 Hasil Pengujian Tahap Kedua Skenario 2

Pengujian tahap kedua skenario 2 adalah pengujian terhadap performa *jitter* dan *packet loss* pada kontroler *Ryu* dan *Pyretic* dengan menggunakan batas *bandwidth* 1024 Mbps. Pengujian performa *jitter* dan *packet loss* menggunakan 9 *client* terhadap 1 *server* dan masing-masing diuji sebanyak lima kali secara sekuensial ditiap kontroler. Kontroler pertama yang diuji adalah *Ryu* dan dilanjutkan pengujian terhadap kontroler *Pyretic* setelahnya. Penggunaan batasan *bandwidth* dilakukan dengan berdasarkan dokumentasi pengujian Iperf. Batasan *bandwidth* digunakan untuk skenario 2 adalah 1024 Mbps. Penjelasan ringkas pada pengujian tahap ini dapat dilihat seperti pada tabel 4.21 berikut:

Tabel 4. 21 Rangkuman pengujian tahap kedua skenario 2

Paramater Uji	Protokol	*Bandwidth Size*	Kontroler	Aplikasi Kontroler	Waktu Pengujian	Jumlah Pengujian
Jitter dan *Packet loss*	UDP	1024 Mbps	*Ryu* dan *Pyretic*	*Non-weighted Dijkstra*	10 detik tiap parameter	5 kali per kontroler

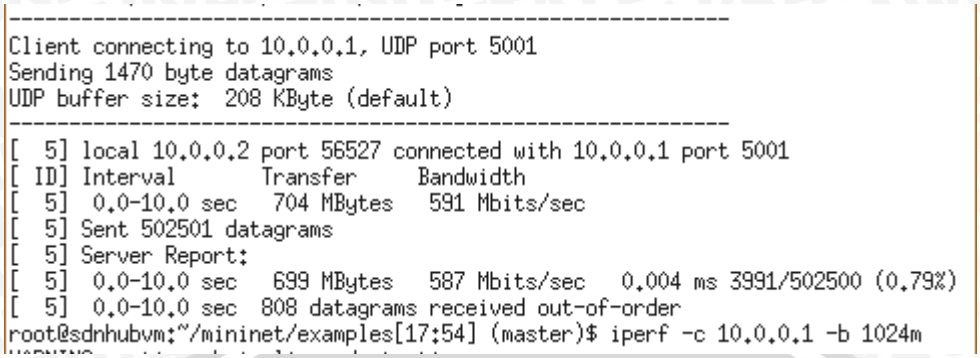
Pengujian tahap kedua skenario 2 :

Skenario 2 : Pengujian *jitter* dan *packet loss* dengan menggunakan parameter batas *bandwidth* 1024 Mbps. Perintah pada iperf seperti pada tabel 4.22 :

Tabel 4. 22 Perintah iperf pada pengujian tahap kedua skenario 2

Server	*Client*
Iperf –s –u	Iperf –c 10.0.0.1 –b 1024m

Pengujian performa *jitter* dan *packet loss* dilakukan dengan menggunakan *batas bandwidth* sebesar 1024 Mbps. Untuk pengujian dilakukan dengan perintah Iperf seperti pada tabel 4.22 dan salah satu hasilnya dapat dilihat seperti pada gambar 4.9.



Gambar 4. 9 Hasil pengujian *throughput* dan *bandwidth* kontroler *Ryu host* 2 ke *server* skenario 2

4.6.1 *Jitter* dan *Packet loss* Pada *Ryu* skenario 2

Pada tahap ini pengujian dilakukan terhadap performa *jitter* dan *packet loss* pada kontroler *Ryu* dengan menggunakan waktu selama 10 detik dan pengujian menggunakan protokol UDP. Iperf digunakan dengan membuat satu *server* dan 9 *client* untuk uji *jitter.* Pengujian menggunakan nilai *bandwidth* sebesar 1024 Mbps. Pengujian dilakukan sebanyak lima kali secara sekuensial. Dan hasil pengujian dirangkum seperti pada tabel 4.23 berikut :

Tabel 4. 23 Hasil pengukuran *jitter* dan *packet loss Ryu* pada batas *bandwidth* 1024 Mbps

Host	Tujuan	Waktu	*Jitter* (ms)	*Packet loss* (%)	Jalur / Path
2	1	10s	0,045	1,280	2-8-7-3-1
3	1	10s	0,028	1,040	3-1
4	1	10s	0,020	1,206	4-6-1
5	1	10s	0,027	1,132	5-7-3-1
6	1	10s	0,022	1,130	6-1
7	1	10s	0,034	1,200	7-3-1
8	1	10s	0,048	1,224	8-7-3-1
9	1	10s	0,051	1,328	9-5-7-3-1
10	1	10s	0,046	1,380	10-9-5-7-3-1

Setelah pengujian dilakukan, hasil dari lima kali pengujian dikalkulasikan menjadi data rata-rata seperti pada tabel 4.23. Pembahasan terhadap hasil pengujian akan dijelaskan pada bab selanjutnya.

4.6.2 *Jitter* dan *Packet loss* Pada *Pyretic* skenario 2

Pada tahap ini pengujian dilakukan terhadap performa *jitter* dan *packet loss* pada kontroler *Pyretic* dengan menggunakan waktu selama 10 detik dan pengujian menggunakan protokol UDP. Iperf digunakan dengan membuat satu *server* dan 9 *client* untuk uji *jitter.* Pengujian menggunakan nilai *bandwidth* sebesar 1024 Mbps. Pengujian dilakukan sebanyak lima kali secara sekuensial. Dan hasil pengujian dirangkum seperti pada tabel 4.24 berikut :

Tabel 4. 24 Hasil pengukuran *jitter* dan *packet loss Pyretic* pada batas *bandwidth* 1024 Mbps

Host	Tujuan	Waktu	*Jitter* (ms)	*Packet loss* (%)	Jalur / Path
2	1	10s	0,043	0,854	2-8-7-3-1
3	1	10s	0,029	0,560	3-1
4	1	10s	0,017	0,556	4-6-1
5	1	10s	0,008	0,564	5-7-3-1
6	1	10s	0,021	0,542	6-1
7	1	10s	0,023	0,586	7-3-1
8	1	10s	0,031	0,608	8-7-3-1
9	1	10s	0,034	0,660	9-5-7-3-1
10	1	10s	0,038	0,820	10-9-5-7-3-1

Setelah pengujian dilakukan, hasil dari lima kali pengujian dikalkulasikan menjadi data rata-rata seperti pada tabel 4.24. Pembahasan terhadap hasil pengujian akan dijelaskan pada bab selanjutnya.

4.7 Hasil Pengujian Tahap Ketiga

Pengujian tahap pertama ketiga adalah pengujian terhadap pemakaian CPU dan RAM dalam pengujian tahap pertama pada kontroler *Ryu* dan *Pyretic*. Pengujian dilakukan dengan cara merekam pemakaian CPU dan RAM ketika kontroler sedang dijalankan. Yang direkam adalah pemakaian CPU dan RAM pada tiap kontroler saat dalam keadaan *idle* selama 10 detik dan keadaan melakukan pengujian *host* 2 pada TCP dan UDP selama 40 detik. Penjelasan ringkas pada pengujian tahap ini dapat dilihat seperti pada tabel 4.25.

Tabel 4. 25 Rangkuman pengujian tahap ketiga

Paramater Uji	Kontroler	Uji QoS	Waktu Pengujian	Jumlah Pengujian
Pemakaian CPU dan RAM (%)	*Ryu* dan *Pyretic*	*throuhput*	60 detik 10 detik *idle,* 40 detik pengujian QoS, 10 detik *idle*	5 kali per kontroler

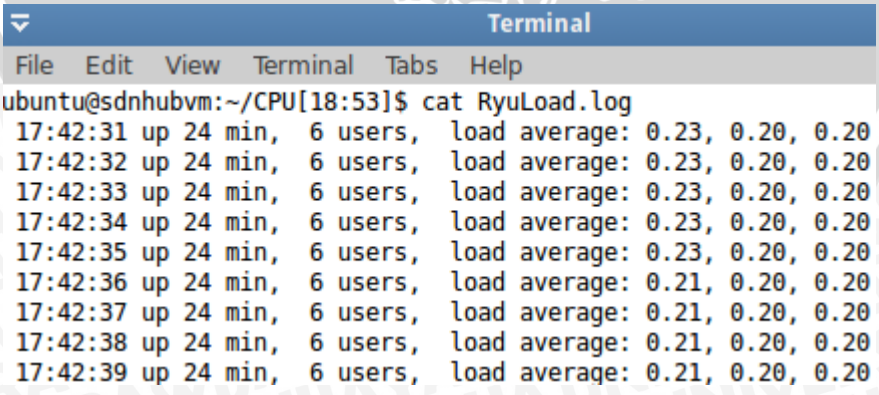
Pengujian tahap ketiga : Pengujian terhadap pemakaian CPU dan RAM dua kontroler ketika dalam keadaan *idle* dan keadaan pengujian *TCP* dan *UDP*. Perintah untuk melakukan perekaman aktivitas CPU dan RAM ditunjukkan seperti pada tabel 4.26 berikut :

Tabel 4. 26 Perintah *CLI (shell*) untuk memonitor beban CPU perdetik

Perintah
while true; do uptime >> <nama_log>.log; sleep 1; done

4.7.1 Pemakaian CPU *Ryu*

Pada tahap ini pengujian dilakukan pada kontroler *Ryu* dengan menggunakan waktu selama satu menit. Pengujian kontroler *Ryu* menggunakan Iperf seperti pada pengujian tahap pertama. Perekaman pemakaian CPU dimulai saat kontroler mulai dijalankan. Perekaman pemakaian CPU dilakukan selama satu menit diawali dengan merekam dalam keadaan *idle* selama 10 detik pertama dan dilanjutkan dengan merekam pemakaian CPU ketika memulai pengujian QoS pada 40 detik selanjutnya dan merekam pemakaian CPU ketika berhenti melakukan pengujian pada 10 detik terakhir. Berikut hasil perekaman pemakaina CPU seperti pada gambar 4.10 dan pada tabel 4.27:



Gambar 4. 10 Hasil perekaman pemakaian CPU pada kontroler *Ryu*

Data yang diperoleh dalam bentuk data *log* kemudian dimasukkan kedalam bentuk tabel seperti pada tabel 4.25 berikut :

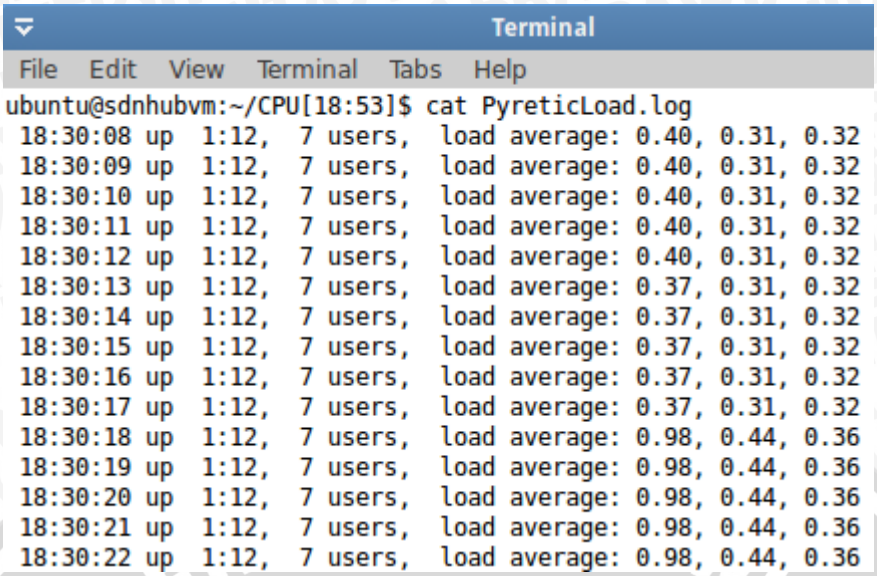
Tabel 4. 27 Hasil perekaman pemakaian CPU kontroler *Ryu* selama satu menit

Waktu ke	10 detik pertama *idle* (%)	10 detik kedua (%)	10 detik ketiga (%)	10 detik keempat (%)	10 detik kelima (%)	10 detik keeenam *idle* (%)
1	14,3	25,8	66,0	63,9	63,6	25,5
2	14,1	66,0	65,3	66,1	66,3	12,7
3	14,6	64,9	64,8	62,9	63,1	12,5
4	13,4	67,7	65,6	61,6	64,4	12,7
5	14,3	66,7	66,0	65,3	67,3	13,5
6	13,1	68,2	62,5	65,4	66,5	13,6
7	13,3	63,1	66,5	63,4	66,0	13,2
8	12,0	65,4	64,5	65,9	66,3	12,0
9	12,0	68,6	65,3	64,6	66,3	13,3
10	12,4	66,8	65,8	63,6	58,1	10,7
Beban RAM		13,6 %				

setelah pengamatan dilakukan, hasil dari lima kali pengamatan dikalkulasikan menjadi data rata-rata seperti pada tabel 4.27. Pembahasan terhadap hasil pengamatan akan dijelaskan pada bab selanjutnya.

4.7.2 Pemakaian CPU *Pyretic*

Pada tahap ini pengujian dilakukan pada kontroler *Pyretic* dengan menggunakan waktu selama satu menit. Pengujian kontroler *Pyretic* menggunakan Iperf seperti pada pengujian tahap pertama. Perekaman pemakaian CPU dan RAM dimulai saat kontroler mulai dijalankan. Perekaman pemakaian CPU dan RAM dilakukan selama satu menit diawali dengan merekam dalam keadaan *idle* selama 10 detik pertama dan dilanjutkan dengan merekam pemakaian CPU dan RAM ketika memulai pengujian QoS pada 40 detik selanjutnya dan merekam pemakaian CPU dan RAM ketika berhenti melakukan pengujian pada 10 detik terakhir. Berikut hasil perekaman pemakaian CPU dan RAM seperti pada gambar 4.11 dan pada tabel 4.28.



Gambar 4. 11 Hasil perekaman pemakaian CPU dan RAM pada kontroler *Pyretic*

data yang diperoleh dalam bentuk data *log* kemudian data rata-rata dimasukkan kedalam bentuk tabel seperti pada tabel 4.28 berikut :

Tabel 4. 28 Hasil perekaman pemakaian CPU dan RAM kontroler *Ryu*

Waktu ke	10 detik pertama *idle* (%)	10 detik kedua (%)	10 detik ketiga (%)	10 detik keempat (%)	10 detik kelima (%)	10 detik keeenam *idle* (%)
1	10,0	25,1	61,4	61,1	59,9	29,4
2	8,0	50,2	62,9	60,3	58,8	10,3
3	7,6	61,3	61,8	58,6	58,8	5,6
4	7,0	62,8	59,0	57,2	60,4	7,7
5	8,6	64,0	59,6	60,9	59,7	8,6
6	7,8	68,0	60,4	60,9	60,6	7,9
7	8,1	61,1	61,1	59,8	62,2	6,7
8	8,3	60,5	58,6	59,0	62,3	7,9
9	7,0	58,9	57,1	58,7	53,3	6,0
10	10,1	60,4	59,7	60,6	35,6	8,0
Beban RAM		12,6 %				

setelah pengamatan dilakukan, hasil dari lima kali pengamatan dikalkulasikan menjadi data rata-rata seperti pada tabel 4.28. Pembahasan terhadap hasil pengamatan akan dijelaskan pada bab selanjutnya.

BAB 5 PEMBAHASAN

Pada bab ini membahas tentang analisis seluruh hasil pengujian yang dilakukan pada bab sebelumnya. Pada bab ini analisis dilakukan menjadi enam sub bab. Sub bab tersebut adalah analisis perbandingan *throughput* dan *bandwidth* kontroler *Ryu* dan *Pyretic* pada skenario 1, analisis perbandingan *throughput* dan *bandwidth* kontroler *Ryu* dan *Pyretic* pada skenario 2, analisis perbandingan *throughput* dan *bandwidth* kontroler *Ryu* dan *Pyretic* pada skenario 3, analisis perbandingan *jitter* dan *packet loss* kontroler *Ryu* dan *Pyretic* pada skenario 1, analisis perbandingan *jitter* dan *packet loss* kontroler *Ryu* dan *Pyretic* pada skenario 2, dan analisis perbandingan pemakaian CPU dan RAM pada kontroler *Ryu* dan *Pyretic.*

5.1 Analisis

Pada sub bab ini akan diberikan penjelasan dan analisis terhadap hasil pengujian yang telah dijelaskan sebelumnya. Analisis meliputi perbandingan hasil pengujian terhadap kedua kontroler dengan parameter yang telah ditentukan. Analisis ini meliputi hasil pengujian *throughput, bandwidth, Jitter, packet loss,* dan pemakaian CPU pada kedua kontroler. Analisis disajikan dengan memberikan data tabel yang membandingkan hasil pengujian pada dua kontroler secara langsung. Analisis juga menjabarkan setiap hasil dari perbandingan kedua kontroler.

5.1.1 Analisis Perbandingan *Throughput* Dan *Bandwidth* Pada *Ryu* Dan *Pyretic* Skenario 1

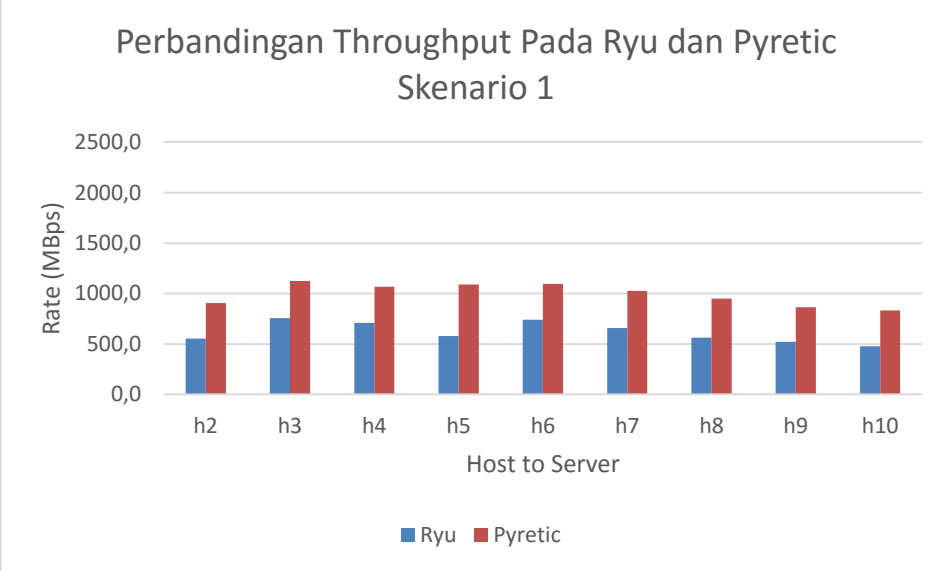
Analisis perbandingan *throughput* dan *bandwidth* dibagi menjadi dua yaitu Analisis perbandingan *throughput* pada *Ryu Dan Pyretic* skenario 1 dan Analisis perbandingan *bandwidth* pada *Ryu* Dan *Pyretic* skenario 2. Analisis pada skenario 1 ini merupakan analisis yang membandingkan *throughput* dan *bandwidth* dengan ukuran *bandwidth* sebesar 100 Mbps dan pada skenario 2 menggunakan *bandwidth* 1024 Mbps.

5.1.1.1 Analisis Perbandingan *Throughput* Pada *Ryu* Dan *Pyretic Skenario 1*

Analisis perbandingan berikut ini adalah perbandingan hasil *throughput* pada kontroler *Ryu* dan *Pyretic* setelah dilakukan pengujian sesuai dengan skenario. Data hasil pengujian dimasukkan dan dibandingkan dalam tabel seperti pada tabel 5.1 dan data hasil pengujian tersebut ditampilkan dalam bentuk grafik seperti pada gambar 5.1.

Tabel 5. 1 Perbandingan *throughput* pada *Ryu* dan *Pyretic* skenario 1

Throughput				
Host	**Tujuan**	***Ryu (MBps)***	***Pyretic (MBps)***	**Jalur / Path**
2	1	553,9	905,8	2-8-7-3-1
3	1	757,2	1125,1	3-1
4	1	708,6	1067,8	4-6-1
5	1	578,6	1088,6	5-7-3-1
6	1	739,8	1096,9	6-1
7	1	657,3	1026,1	7-3-1
8	1	562,8	950,3	8-7-3-1
9	1	520,3	864,5	9-5-7-3-1
10	1	477,3	832,6	10-9-5-7-3-1



Gambar 5. 1 Perbandingan *throughput* pada *Ryu* dan *Pyretic* skenario 1

Tabel 5.1 dan gambar 5.1 menggambarkan rata-rata nilai *throughput* pada skenario 1. Rata-rata *throughput* ini dikalkulasi setelah lima kali pengujian. Pengujian ini dilakukan dengan menguji *host* 2 sampai *host* 10 terhadap *server*

menggunakan Iperf untuk menguji performa *throughput* kontroler *Ryu*. Pengujian dilakukan secara sekuensial.

Berdasarkan tabel 5.1 dan gambar 5.1, rata-rata nilai *throughput* kontroler *Ryu* pada *host* 2 mencapai nilai 553,9 MBps sedangkan rata-rata nilai *throughput* kontroler *Pyretic* lebih tinggi yakni 905,8 MBps pada jalur transmisi 2-8-7-3-1. Hal yang sama seperti perbandingan pada *host* 2 juga terjadi pada *host* 3 sampai *host* 10. Perbandingan rata-rata nilai *throughput* pada *host* 3 adalah 757,2 MBps untuk *Ryu* dan 1125,1 MBps untuk *Pyretic* dengan jalur 3-1. Perbandingan rata-rata nilai *throughput* pada *host* 4 adalah 708,6 MBps untuk *Ryu* dan 1067,8 MBps untuk *Pyretic* dengan jalur 4-6-1. Perbandingan rata-rata nilai *throughput* pada *host* 5 adalah 578,6 MBps untuk *Ryu* dan 1088,6 MBps untuk *Pyretic* dengan jalur 5-7-3-1. Perbandingan rata-rata nilai *throughput* pada *host* 6 adalah 739,8 MBps untuk *Ryu* dan 1096,9 MBps untuk *Pyretic* dengan jalur 6-1. Perbandingan rata-rata nilai *throughput* pada *host* 7 adalah 657,3 MBps untuk *Ryu* dan 1026,1 MBps untuk *Pyretic* dengan jalur 7-3-1. Perbandingan rata-rata nilai *throughput* pada *host* 8 adalah 562,8 MBps untuk *Ryu* dan 950,3 MBps untuk *Pyretic* dengan jalur 8-7-3-1. Perbandingan rata-rata nilai *throughput* pada *host* 9 adalah 520,3 MBps untuk *Ryu* dan 864,5 MBps untuk *Pyretic* dengan jalur 9-5-7-3-1. Perbandingan rata-rata nilai *throughput* pada *host* 10 adalah 477,3 MBps untuk *Ryu* dan 832,6 MBps untuk *Pyretic* dengan jalur 10-9-5-7-3-1.

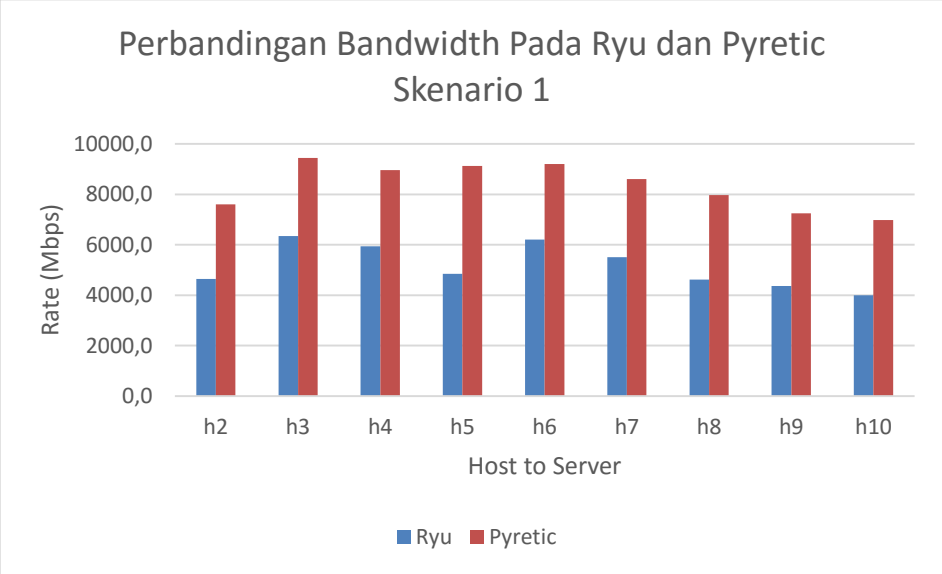
Secara keseluruhan rata-rata *throughput* pada kontroler *Pyretic* lebih tinggi dibanding dengan nilai *throuhput* pada kontroler *Ryu*. Nilai tertinggi pada kontroler *Pyretic* terjadi pada *host 3* dengan nilai 1125,1 MBps dengan jalur 3-1, sedangkan pada *host* yang sama kontroler *Pyretic* memiliki nilai *throughput* sebesar 757,2 MBps. Nilai *throughput* pada kedua kontroler terpengaruh pada jalur transmisi yang dilalui. Jika jalur transmisi menuju *server* semakin panjang atau semakin banyak switch maka nilai *throughput* akan semakin rendah.

5.1.1.2 Analisis Perbandingan *Bandwidth* Pada *Ryu* Dan *Pyretic Skenario 1*

Analisis perbandingan berikut ini adalah perbandingan hasil *bandwidth* pada kontroler *Ryu* dan *Pyretic* setelah dilakukan pengujian sesuai dengan skenario. Data hasil pengujian dimasukkan dalam tabel seperti pada tabel 5.2 dan data hasil pengujian tersebut ditampilkan dalam bentuk grafik seperti pada gambar 5.2.

Tabel 5. 2 Perbandingan *bandwidth* pada *Ryu* dan *Pyretic* skenario 1

Bandwidth				
Host	**Tujuan**	***Ryu (Mbps)***	***Pyretic (Mbps)***	**Jalur / Path**
2	1	4645,8	7598,4	2-8-7-3-1
3	1	6351,4	9438,0	3-1
4	1	5943,6	8957,2	4-6-1
5	1	4853,6	9131,8	5-7-3-1
6	1	6204,8	9208,2	6-1
7	1	5513,0	8607,4	7-3-1
8	1	4620,0	7971,8	8-7-3-1
9	1	4364,0	7251,4	9-5-7-3-1
10	1	4004,0	6985,0	10-9-5-7-3-1



Gambar 5. 2 Perbandingan *bandwidth* pada *Ryu* dan *Pyretic* skenario 1

Tabel 5.2 dan gambar 5.2 menggambarkan rata-rata nilai *bandwidth* pada skenario 1. Rata-rata *bandwidth* ini dikalkulasi setelah lima kali pengujian. Pengujian ini dilakukan dengan menguji *host* 2 sampai *host* 10 terhadap *server* menggunakan Iperf untuk menguji performa *bandwidth* kontroler *Ryu*. Pengujian dilakukan secara sekuensial.

Berdasarkan tabel 5.2 dan gambar 5.2, hasil perbandingan *bandwidth* pada kedua kontroler hampir sama dengan perbandingan *throughput* karena pengujian dilakukan secara bersamaan namun nilai kedua parameter berbeda. Rata-rata

nilai *bandwidth* kontroler *Ryu* pada *host* 2 mencapai nilai 4645,8 Mbps sedangkan rata-rata nilai *throughput* kontroler *Pyretic* lebih tinggi yakni 7598,4 Mbps pada jalur transmisi 2-8-7-3-1. Perbandingan rata-rata nilai *bandwidth* pada *host* 3 adalah 6351,4 Mbps untuk *Ryu* dan 9438,0 Mbps untuk *Pyretic* dengan jalur 3-1. Perbandingan rata-rata nilai *bandwidth* pada *host* 4 adalah 5943,6 Mbps untuk *Ryu* dan 8957,2 Mbps untuk *Pyretic* dengan jalur 4-6-1. Perbandingan rata-rata nilai *bandwidth* pada *host* 5 adalah 4853,6 Mbps untuk *Ryu* dan 9131,8 Mbps untuk *Pyretic* dengan jalur 5-7-3-1. Perbandingan rata-rata nilai *bandwidth* pada *host* 6 adalah 6204,8 Mbps untuk *Ryu* dan 9208,2 Mbps untuk *Pyretic* dengan jalur 6-1. Perbandingan rata-rata nilai *bandwidth* pada *host* 7 adalah 5513 Mbps untuk *Ryu* dan 8607,4 Mbps untuk *Pyretic* dengan jalur 7-3-1. Perbandingan rata-rata nilai *bandwidth* pada *host* 8 adalah 4620 Mbps untuk *Ryu* dan 7971,8 Mbps untuk *Pyretic* dengan jalur 8-7-3-1. Perbandingan rata-rata nilai *bandwidth* pada *host* 9 adalah 4364,0 Mbps untuk *Ryu* dan 7251,4 Mbps untuk *Pyretic* dengan jalur 9-5-7-3-1. Perbandingan rata-rata nilai *bandwidth* pada *host* 10 adalah 4004 Mbps untuk *Ryu* dan 6985 Mbps untuk *Pyretic* dengan jalur 10-9-5-7-3-1.

Secara keseluruhan rata-rata *bandwidth* pada kontroler *Pyretic* lebih tinggi dibanding dengan nilai *throuhput* pada kontroler *Ryu*. Nilai tertinggi pada kontroler *Ryu* terjadi pada *host 3* dengan nilai 6351,4 Mbps dengan jalur 3-1, sedangkan pada *host* yang sama kontroler *Pyretic* memiliki nilai *bandwidth* sebesar 9438 Mbps. Nilai *bandwidth* pada kedua kontroler juga terpengaruh pada jalur transmisi yang dilalui sama seperti *throughput*. Jika jalur transmisi menuju *server* semakin panjang atau semakin banyak switch maka nilai *bandwidth* akan semakin rendah.

5.1.2 Analisis Perbandingan *Throughput* Dan *Bandwidth* Pada *Ryu* Dan *Pyretic* Skenario 2

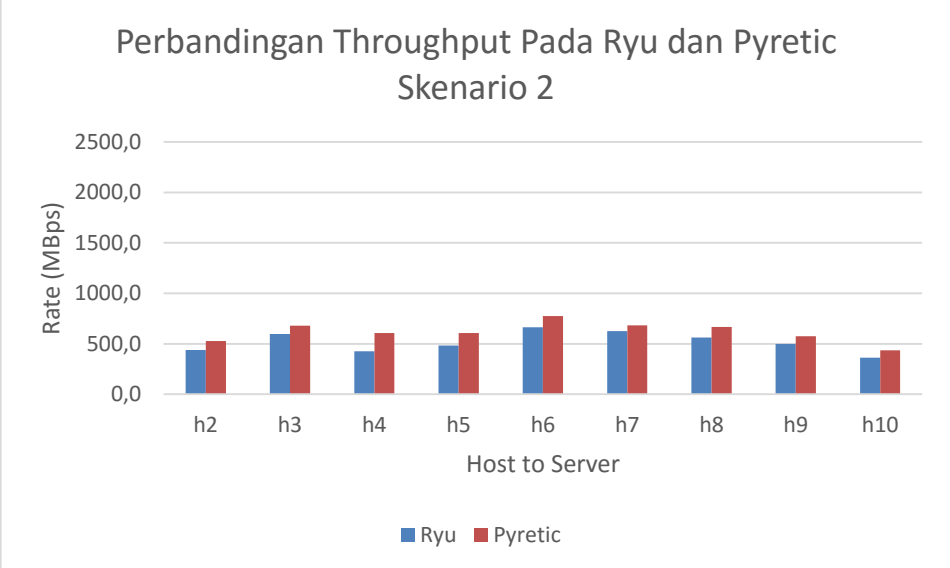
Analisis perbandingan *throughput* dan *bandwidth* dibagi menjadi dua yaitu Analisis Perbandingan *throughput* Pada *Ryu Dan Pyretic* Skenario 2 dan Analisis Perbandingan *bandwidth* Pada *Ryu* Dan *Pyretic* Skenario 2. Pada analisis ini akan dijabarkan perbandingan *throughput* dan *bandwidth* antara dua kontroler secara langsung. Perandingan tersebut menggunakan skenario 2 yang artinya ukuran *TCP window* sebesar 170 KB.

5.1.2.1 Analisis Perbandingan *Throughput* Pada *Ryu* Dan *Pyretic Skenario 2*

Analisis perbandingan berikut ini adalah perbandingan hasil *throughput* pada kontroler *Ryu* dan *Pyretic* setelah dilakukan pengujian sesuai dengan skenario 2. Data hasil pengujian dimasukkan dalam tabel seperti pada tabel 5.3 dan data hasil pengujian tersebut ditampilkan dalam bentuk grafik seperti pada gambar 5.3

Tabel 5. 3 Perbandingan *Throughput* Pada *Ryu* Dan *Pyretic* Skenario 2

Throughput				
Host	**Tujuan**	***Ryu (MBps)***	***Pyretic (MBps)***	**Jalur / Path**
2	1	439,8	526,0	2-8-7-3-1
3	1	596,6	679,5	3-1
4	1	426,2	606,3	4-6-1
5	1	483,6	607,4	5-7-3-1
6	1	664,5	773,5	6-1
7	1	624,7	682,3	7-3-1
8	1	561,6	665,6	8-7-3-1
9	1	499,8	575,9	9-5-7-3-1
10	1	362,3	434,3	10-9-5-7-3-1



Gambar 5. 3 Perbandingan *throughput* pada *Ryu* dan *Pyretic* skenario 2

Berdasarkan tabel 5.4 dan gambar 5.4, dapat disimpulkan bahwa nilai rata-rata *throughput* kontroler *Pyretic* masih lebih unggul dari nilai *throughput* kontroler *Ryu.* Hasil nilai rata-rata *throughput* pada skenario 2 ini dipengaruhi oleh ukuran *window size* yang lebih besar dari 85,5 KB yang digunakan pada skenario 1. Hasilnya menunjukkan bahwa penggunaan ukuran TCP lebih besar tidak memberikan nilai *throughput* lebih besar pula.

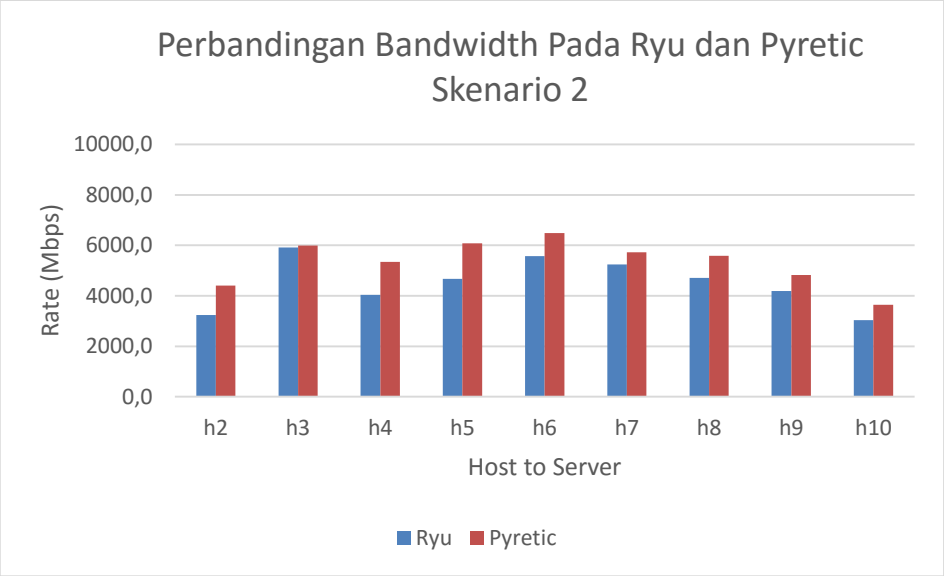
Nilai rata-rata tertinggi juga terjadi pada *host* ke 3 dengan nilai rata-rata *throughput* sebesar 596,6 MBps pada kontroler *Ryu* dan 679,5 MBps dengan jarak terpendek yaitu 3-1. Dan nilai rata-rata *throughput* terendah berada pada *host* 10 dengan nilai rata-rata *throughput* pada kontroker *Ryu* sebesar 362,3 MBps dan 434,3 MBps pada kontroler *Pyretic* dengan jalur terjauh yaitu 10-9-5-7-3-1. Hasil *throughput* sama halnya dengan skenario 1 memiliki pengaruh pada panjang jalur transmisi. Semakin jauh jalur transmisi maka nilai rata-rata *throughput* juga akan semakin rendah.

5.1.2.2 Analisis Perbandingan *Bandwidth* Pada *Ryu* Dan *Pyretic Skenario 2*

Analisis perbandingan berikut ini adalah perbandingan hasil *bandwidth* pada kontroler *Ryu* dan *Pyretic* setelah dilakukan pengujian sesuai dengan skenario 2. Data hasil pengujian dimasukkan dalam tabel seperti pada tabel 5.4 dan data hasil pengujian tersebut ditampilkan dalam bentuk grafik seperti pada gambar 5.4 berikut :

Tabel 5. 4 Perbandingan *bandwidth* pada *Ryu dan Pyretic* skenario 2

Bandwidth				
Host	**Tujuan**	***Ryu (Mbps)***	***Pyretic (Mbps)***	**Jalur / Path**
2	1	3243,2	4412,0	2-8-7-3-1
3	1	5919,6	5996,6	3-1
4	1	4033,0	5350,0	4-6-1
5	1	4675,2	6081,6	5-7-3-1
6	1	5574,0	6488,2	6-1
7	1	5240,2	5724,0	7-3-1
8	1	4710,6	5583,0	8-7-3-1
9	1	4192,8	4830,8	9-5-7-3-1
10	1	3039,6	3642,8	10-9-5-7-3-1



Gambar 5. 4 Perbandingan *bandwidth* pada *Ryu* dan *Pyretic* skenario 2

Berdasarkan tabel 5.4 dan gambar 5.4, dapat disimpulkan bahwa nilai rata-rata *throughput* kontroler *Pyretic* masih lebih unggul dari nilai *bandwidth* kontroler *Ryu.* Hasil nilai rata-rata *bandwidth* pada skenario 2 ini dipengaruhi oleh ukuran TCP *window size* yang lebih besar dari 85,5 KB yaitu 170 KB. Hasilnya menunjukkan bahwa penggunaan ukuran TCP lebih besar tidak memberikan nilai *bandwidth* lebih besar.

Nilai rata-rata tertinggi juga terjadi pada *host* ke 3 dengan nilai rata-rata *bandwidth* sebesar 5919,6 Mbps pada kontroler *Ryu* dan 5996,6 Mbps dengan jarak terpendek. Dan nilai rata-rata *bandwidth* terendah berada pada *host* 10 dengan nilai rata-rata *bandwidth* pada kontroler *Ryu* sebesar 3039,6 Mbps dan 3642,8 Mbps pada kontroler *Pyretic* dengan jalur terjauh yaitu 10-9-5-7-3-1. Hasil *bandwidth* sama halnya dengan skenario 1 memiliki pengaruh pada panjang jalur transmisi. Semakin jauh jalur transmisi maka nilai rata-rata *bandwidth* juga akan semakin rendah.

5.1.3 Analisis Perbandingan *Throughput* Dan *Bandwidth* Pada *Ryu* Dan *Pyretic* Skenario 3

Analisis perbandingan *throughput* dan *bandwidth* dibagi menjadi dua yaitu Analisis Perbandingan *throughput* Pada *Ryu Dan Pyretic* Skenario 3 dan Analisis Perbandingan *bandwidth* Pada *Ryu* Dan *Pyretic* Skenario 3. Pada analisis ini dijabarkan perbandingan *throughput* dan *bandwidth* secara langsung. Pada analisis ini juga digunakan skenario 3 yang artinya ukuran *TCP window* sebesar 42 KB.

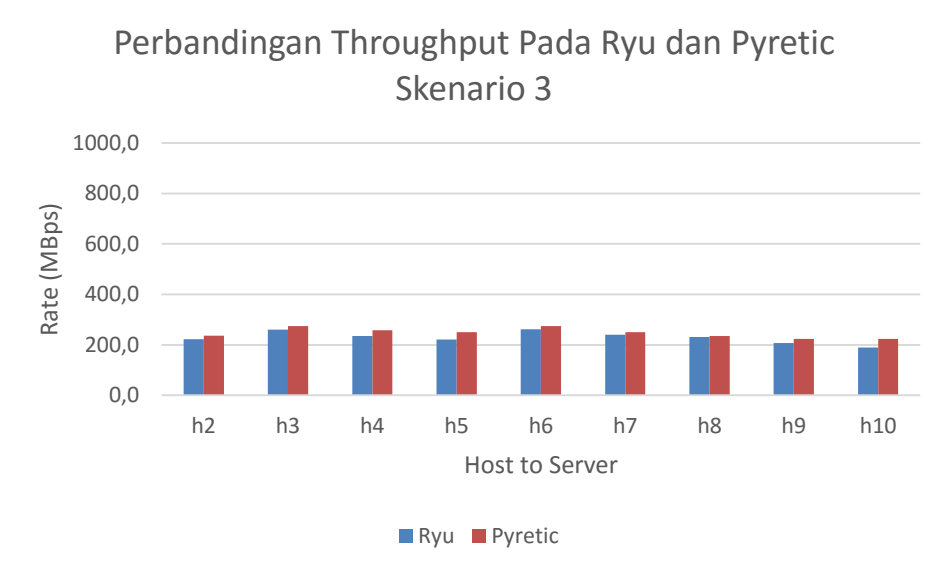
5.1.3.1 Analisis Perbandingan *Throughput* Pada *Ryu* Dan *Pyretic Skenario 3*

Perbandingan nilai rata-rata *throughput* seperti pada tabel 5.5 berdasarkan hasil pengujian yang telah dilakukan dengan menggunakan TCP

window size sebasar 42 KB. Berikut hasil perbandingan *throughput* pada kedua kontroler seperti pada tabel 5.5.

Tabel 5. 5 Perbandingan *throughput* pada *Ryu* dan *Pyretic* skenario 3

Throughput				
Host	**Tujuan**	***Ryu (MBps)***	***Pyretic (MBps)***	**Jalur / Path**
2	1	222,4	236,4	2-8-7-3-1
3	1	260,8	274,4	3-1
4	1	234,5	258,4	4-6-1
5	1	221,3	250,6	5-7-3-1
6	1	261,9	274,3	6-1
7	1	240,3	250,1	7-3-1
8	1	231,7	235,6	8-7-3-1
9	1	207,5	223,7	9-5-7-3-1
10	1	189,7	223,3	10-9-5-7-3-1



Gambar 5. 5 Perbandingan *throughput pada Ryu* dan *Pyretic* skenario 3

Berdasarkan tabel 5.5 dan gambar 5.5, dapat disimpulkan bahwa nilai rata-rata *throughput* kontroler *Pyretic* masih lebih unggul dari nilai *throughput* kontroler *Ryu.* Hasil nilai rata-rata *throughput* pada skenario 2 ini dipengaruhi oleh ukuran *window size* yang lebih kecil dari 85,5 yang digunakan pada skenario 1 yaitu sebesar 42 KB. Hasilnya menunjukkan bahwa penggunaan ukuran TCP lebih kecil juga tidak memberikan nilai *throughput* lebih besar.

Nilai rata-rata tertinggi juga terjadi pada *host* ke 3 dengan nilai rata-rata *throughput* sebesar 260,8 MBps pada kontroler *Ryu* dan 274,4 MBps dengan jarak terpendek yaitu 3-1. Dan nilai rata-rata *throughput* terendah berada pada *host* 10 dengan nilai rata-rata *throughput* pada kontroker *Ryu* sebesar 189,7 MBps dan 223,3 MBps pada kontroler *Pyretic* dengan jalur terjauh yaitu 11-2-8-7-3-1. Hasil *throughput* sama halnya dengan skenario 1 memiliki pengaruh pada panjang jalur transmisi. Semakin jauh jalur transmisi maka nilai rata-rata *throughput* juga akan semakin rendah.

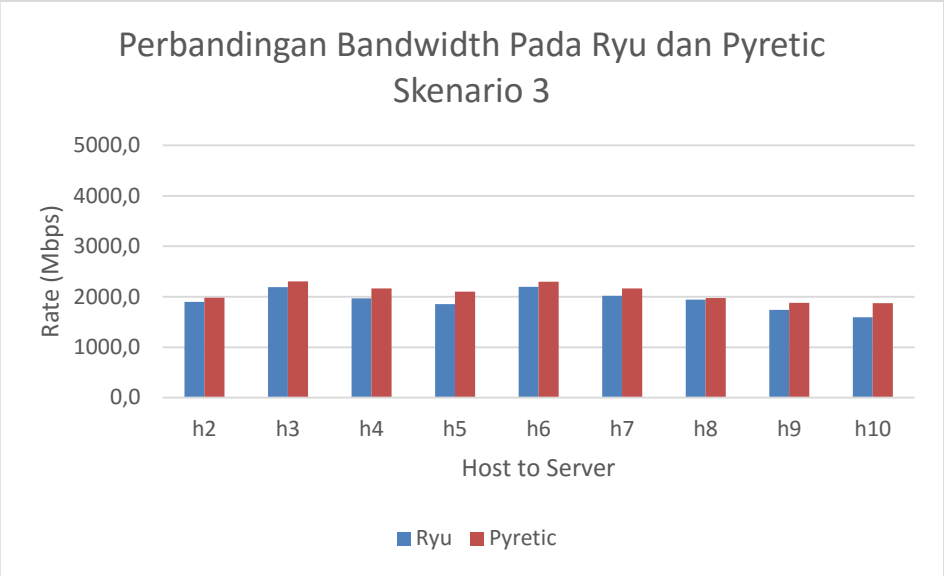
5.1.3.2 Analisis Perbandingan *Bandwidth* Pada *Ryu* Dan *Pyretic* Skenario 3

Perbandingan nilai rata-rata *bandwidth* seperti pada tabel 5.5 berdasarkan hasil pengujian yang telah dilakukan dengan menggunakan TCP *window size* sebasar 42 KB. Berikut hasil perbandingan *bandwidth* pada kedua kontroler seperti pada tabel 5.5 :

Tabel 5. 6 Analisis perbandingan *bandwidth pada Ryu dan Pyretic* skenario 3

Bandwidth				
Host	**Tujuan**	***Ryu (Mbps)***	***Pyretic (Mbps)***	**Jalur / Path**
2	1	1897,6	1983,4	2-8-7-3-1
3	1	2187,2	2301,8	3-1
4	1	1967,2	2167,6	4-6-1
5	1	1856,8	2102,4	5-7-3-1
6	1	2196,6	2300,8	6-1
7	1	2016,2	2162,4	7-3-1
8	1	1943,2	1976,2	8-7-3-1
9	1	1740,4	1876,0	9-5-7-3-1
10	1	1591,6	1873,0	10-9-5-7-3-1

Hasil pengujian seperti pada tabel 5.6 memberikan data rata-rata dari lima kali percobaan pengujian. Data dari seluruh pengujian dirata-rata menjadi data tunggal. Data tunggal tersebut yang dimasukkan kedalam tabel. Data pada tabel terebut kemudian ditampilkan pada grafik yang mudah untuk dibaca seperti pada gambar 5.6



Gambar 5. 6 Perbandingan *bandwidth* pada *Ryu* dan *Pyretic* skenario 3

Berdasarkan tabel 5.6 dan gambar 5.6, dapat disimpulkan bahwa nilai rata-rata *throughput* kontroler *Pyretic* masih lebih unggul dari nilai *bandwidth* kontroler *Ryu.* Hasil nilai rata-rata *bandwidth* pada skenario 3 ini dipengaruhi oleh ukuran TCP *window size* yang lebih besar dari 85,5 KB yang digunakan pada skenario 1 yaitu 42 KB. Hasilnya menunjukkan bahwa penggunaan ukuran TCP lebih besar tidak memberikan nilai *bandwidth* lebih besar.

Nilai rata-rata tertinggi juga terjadi pada *host* ke 3 dengan nilai rata-rata *bandwidth* sebesar 2187,2 Mbps pada kontroler *Ryu* dan 2301,8 Mbps dengan jarak terpendek yaitu 3-1. Dan nilai rata-rata *bandwidth* terendah berada pada *host* 10 dengan nilai rata-rata *bandwidth* pada kontroler *Ryu* sebesar 1591,6 Mbps dan 1873 Mbps pada kontroler *Pyretic* dengan jalur terjauh yaitu 10-9-5-7-3-1. Hasil *bandwidth* sama halnya dengan skenario 1 memiliki pengaruh pada panjang jalur transmisi. Semakin jauh jalur transmisi maka nilai rata-rata *bandwidth* juga akan semakin rendah.

5.1.4 Analisis Perbandingan *Jitter* dan *Packet loss* Pada *Ryu* dan *Pyretic* skenario 1

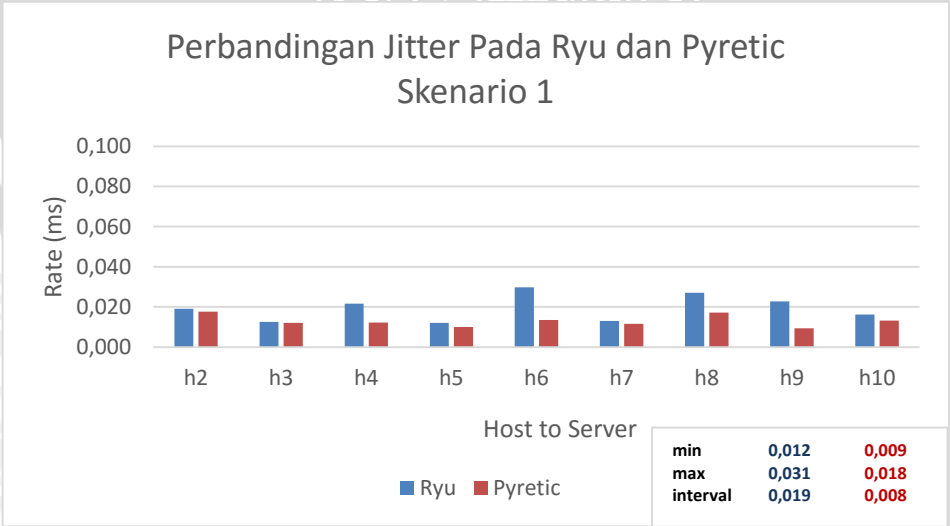
Analisis perbandingan *jitter* dan *packet loss* dibagi menjadi dua yaitu Analisis Perbandingan *jitter* Pada *Ryu Dan Pyretic* Skenario 1 dan Analisis Perbandingan *packet loss* Pada *Ryu* Dan *Pyretic* Skenario 1. Pada analisis ini dijabarkan perbandingan *jitter* dan *packet loss* secara langsung. Pada analisis ini juga digunakan skenario 1 yang artinya ukuran *bandwidth* yang digunakan sebesar 100 Mbps.

5.1.4.1 Analisis Perbandingan *Jitter* Pada *Ryu* Dan *Pyretic Skenario 1*

Perbandingan nilai rata-rata *jitter* seperti pada tabel 5.7 berdasarkan hasil pengujian yang telah dilakukan dengan menggunakan batas *bandwidth* sebasar 100 Mbps. Berikut hasil perbandingan *jitter* pada kedua kontroler seperti pada tabel 5.7.

Tabel 5. 7 Perbandingan *jitter Ryu* dan *Pyretic*

Jitter				
Host	**Tujuan**	***Ryu (ms)***	***Pyretic (ms)***	**Jalur / Path**
2	1	0,019	0,018	2-8-7-3-1
3	1	0,013	0,014	3-1
4	1	0,022	0,012	4-6-1
5	1	0,012	0,010	5-7-3-1
6	1	0,030	0,013	6-1
7	1	0,013	0,012	7-3-1
8	1	0,027	0,017	8-7-3-1
9	1	0,023	0,009	9-5-7-3-1
10	1	0,016	0,013	10-9-5-7-3-1



Gambar 5. 7 Perbandingan *jitter* pada dan *Pyretic* skenario 1

Berdasarkan gambar 5.7 dan tabel 5.7 dapat disimpulkan bahwa secara keseluruhan rata-rata nilai *jitter* pada kontroler *Ryu* lebih tinggi dibanding dengan kontroler *Pyretic*. Kedua kontroler memiliki nilai rata-rata *jitter* yang sangat kecil

dan relatif stabil karena memiliki nilai interval yang sangat kecil. Berdasarkan hasil pengujian *Host* 2 memiliki nilai rata-rata *jitter* 0,019 ms untuk *Ryu* dan 0,018 ms untuk *Pyretic* pada jalur 2-8-3-1. *Host* 3 memiliki nilai rata-rata *jitter* 0,013 ms untuk *Ryu* dan 0,014 ms untuk *Pyretic* pada jalur 3-1. *Host* 4 dengan nilai rata-rata *jitter* 0,022 ms untuk *Ryu* dan 0,012 ms untuk *Pyretic* pada jalur 4-6-1. *Host* 5 dengan nilai rata-rata *jitter* 0,012 ms untuk *Ryu* dan 0,010 ms untuk *Pyretic* pada jalur 5-7-3-1. *Host* 6 dengan nilai rata-rata *jitter* 0,030 ms untuk *Ryu* dan 0,013 ms untuk *Pyretic* pada jalur 6-1. *host* 7 dengan nilai rata-rata *jitter* 0,013 ms untuk *Ryu* dan 0,012 ms untuk *Pyretic* pada jalur 7-3-1. *Host* 8 dengan nilai rata-rata *jitter* 0,027 ms untuk *Ryu* dan 0,017 ms untuk *Pyretic* pada jalur 8-7-3-1. *Host* 9 dengan nilai rata-rata *jitter* 0,023 ms untuk *Ryu* dan 0,009 ms untuk *Pyretic* pada jalur 9-5-7-3-1. Dan *host* 10 dengan nilai rata-rata *jitter* 0,016 ms untuk *Ryu* dan 0,013 ms untuk *Pyretic* pada jalur 10-9-5-7-3-1.

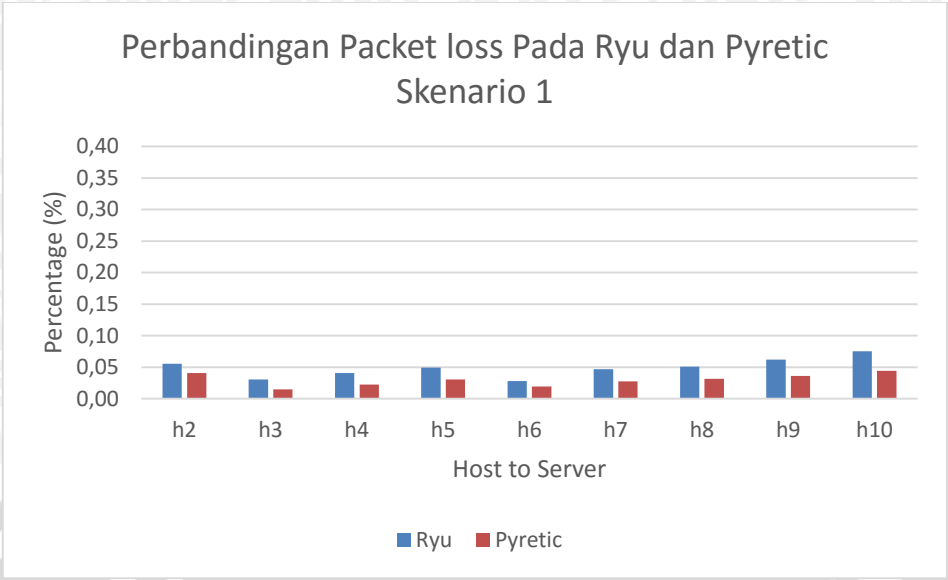
Secara keseluruhan bahwa nilai *jitter* pada kedua kontroler termasuk sangat bagus dalam kategori degradasi dengan nilai rata-rata kurang dari 1 ms. Nilai rata-rata *jitter* pada kontroler *Ryu* lebih tinggi daripada nilai *jitter* pada *Pyretic*. Nilai rata-rata *jitter* kedua kontroler relatif kecil karena selain *bandwidth* 100 Mbps, ukuran paket UDP dalam pengujian 1470 *Byte* sehingga waktu *reassemble* paket memiliki waktu yang cepat.

5.1.4.2 Analisis Perbandingan *Packet loss* Pada *Ryu* Dan *Pyretic* Skenario 1

Perbandingan nilai rata-rata *packet loss* seperti pada tabel 5.7 berdasarkan hasil pengujian yang telah dilakukan dengan menggunakan batas *bandwidth* sebasar 100 Mbps. Berikut hasil perbandingan *jitter* pada kedua kontroler seperti pada tabel 5.7 :

Tabel 5. 8 Analisis perbandingan *packet loss* pada *Ryu* dan *Pyretic* skenario 1

Packet loss				
Host	**Tujuan**	***Ryu* (%)**	***Pyretic* (%)**	**Jalur / Path**
2	1	0,055	0,041	2-8-7-3-1
3	1	0,030	0,015	3-1
4	1	0,041	0,023	4-6-1
5	1	0,049	0,031	5-7-3-1
6	1	0,028	0,019	6-1
7	1	0,047	0,028	7-3-1
8	1	0,051	0,032	8-7-3-1
9	1	0,062	0,036	9-5-7-3-1
10	1	0,075	0,044	10-9-5-7-3-1



Gambar 5. 8 Analisis perbandingan *packet loss* pada *Ryu* dan *Pyretic* skenario 1

Berdasarkan gambar 5.8 dan tabel 5.8 dapat disimpulkan bahwa secara keseluruhan rata-rata nilai *packet loss* pada kontroler *Ryu* lebih tinggi dibanding dengan kontroler *Pyretic*. Berdasarkan hasil pengujian, *Host* 2 memiliki nilai rata-rata *packet loss* 0,055 % untuk *Ryu* dan 0,041 % untuk *Pyretic* pada jalur 2-8-3-1. pada *host* 3 dengan nilai rata-rata *packet loss* 0,030 % untuk *Ryu* dan 0,015 % untuk *Pyretic* pada jalur 3-1. *Host* 4 dengan nilai rata-rata *packet loss* 0,041 % untuk *Ryu* dan 0,023 % untuk *Pyretic* pada jalur 4-6-1. *Host* 5 dengan nilai rata-rata *packet loss* 0,049 % untuk *Ryu* dan 0,031 % untuk *Pyretic* pada jalur 5-7-3-1. *Host* 6 dengan nilai rata-rata *packet loss* 0,028 % untuk *Ryu* dan 0,019 % untuk *Pyretic* pada jalur 6-1. *Host* 7 dengan nilai rata-rata *packet loss* 0,047 % untuk *Ryu* dan 0,028 % untuk *Pyretic* pada jalur 7-3-1. *Host* 8 dengan nilai rata-rata *packet loss* 0,051 % untuk *Ryu* dan 0,032 % untuk *Pyretic* pada jalur 8-7-3-1. *Host* 9 dengan nilai rata-rata *packet loss* 0,062 % untuk *Ryu* dan 0,036 % untuk *Pyretic* pada jalur 9-5-7-3-1. *Host* 10 dengan nilai rata-rata *packet loss* 0,075 % untuk *Ryu* dan 0,044 % untuk *Pyretic* pada jalur 10-9-5-7-3-1.

Secara keseluruhan bahwa nilai *packet loss* pada kedua kontroler termasuk sangat bagus dalam kategori degradasi dengan prosentase rata-rata kurang dari 1 %. Nilai rata-rata *packet loss* pada kontroler *Pyretic* lebih rendah daripada nilai *packet loss* pada *Ryu*. Selain batas *bandwidth* yang ditentukan yakni 100 Mbps, hal ini juga dipengaruhi ukuran *buffer* UDP dalam pengujian. Karena ukuran *buffer UDP* sebesar 208 KB maka jumlah *packet loss* masih memiliki persentase yang sangat bagus.

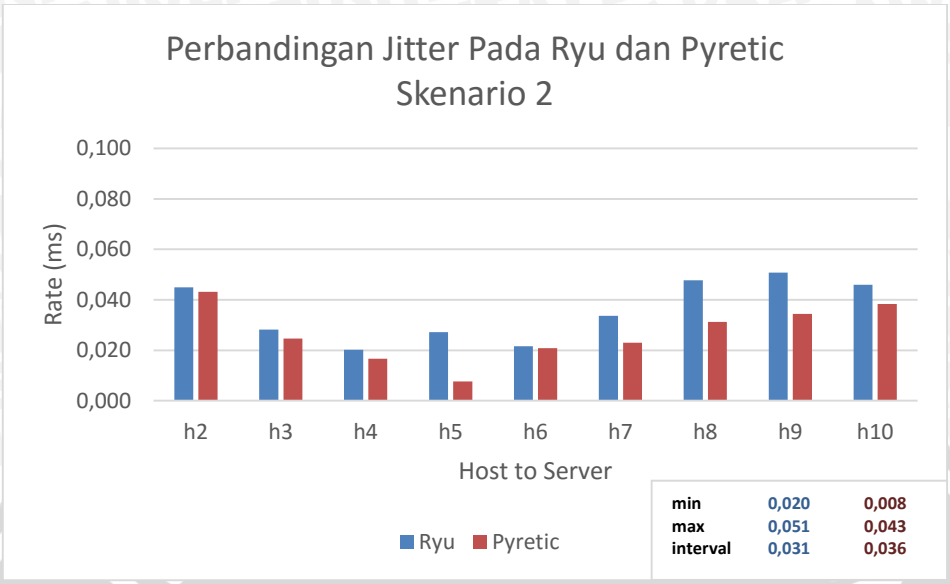
5.1.5 Analisis Perbandingan *Jitter* dan *Packet loss* Pada *Ryu* dan *Pyretic* skenario 2

Analisis perbandingan *jitter* dan *packet loss* dibagi menjadi dua yaitu Analisis Perbandingan *jitter* Pada *Ryu Dan Pyretic* Skenario 2 dan Analisis Perbandingan *packet loss* Pada *Ryu* Dan *Pyretic* Skenario 2. Pada analisis ini dijabarkan perbandingan *jitter* dan *packet loss* secara langsung. Pada analisis ini juga digunakan skenario 1 yang artinya ukuran *bandwidth* yang digunakan sebesar 1024 Mbps.

5.1.5.1 Analisis Perbandingan *Jitter* Pada *Ryu* Dan *Pyretic* Skenario 2

Tabel 5. 9 Perbandingan *jitter Ryu* dan *Pyretic*

Jitter				
Host	**Tujuan**	***Ryu (ms)***	***Pyretic (ms)***	**Jalur / Path**
2	1	0,045	0,043	2-8-7-3-1
3	1	0,028	0,029	3-1
4	1	0,020	0,017	4-6-1
5	1	0,027	0,008	5-7-3-1
6	1	0,022	0,021	6-1
7	1	0,034	0,023	7-3-1
8	1	0,048	0,031	8-7-3-1
9	1	0,051	0,034	9-5-7-3-1
10	1	0,046	0,038	10-9-5-7-3-1



Gambar 5. 9 Perbandingan *jitter* pada *ryu* dan *pyretic skenario 2*

Berdasarkan gambar 5.9 dan tabel 5.9 dapat disimpulkan bahwa secara keseluruhan nilai rata-rata *jitter* pada kontroler *Ryu* masih lebih tinggi dibanding dengan kontroler *Pyretic*. Nilai rata-rata *jitter* tertinggi terjadi pada *host* 9 dengan nilai rata-rata *jitter* 0,051 ms untuk *Ryu* dan 0,034 ms untuk *Pyretic* pada jalur 9-5-7-3-1. Dan terendah pada *host* 5 dengan nilai rata-rata *jitter* 0,027 ms untuk *Ryu* dan 0,008 ms untuk *Pyretic* pada jalur 5-7-3-1. Hasil tersebut masih menunjukkan bahwa nilai rata-rata *jitter* kedua kontroler relatif stabil dengan nilai interval sebesar 0,031 ms pada kontroler *Ryu* dan 0,036 ms pada kontroler *Pyretic.*

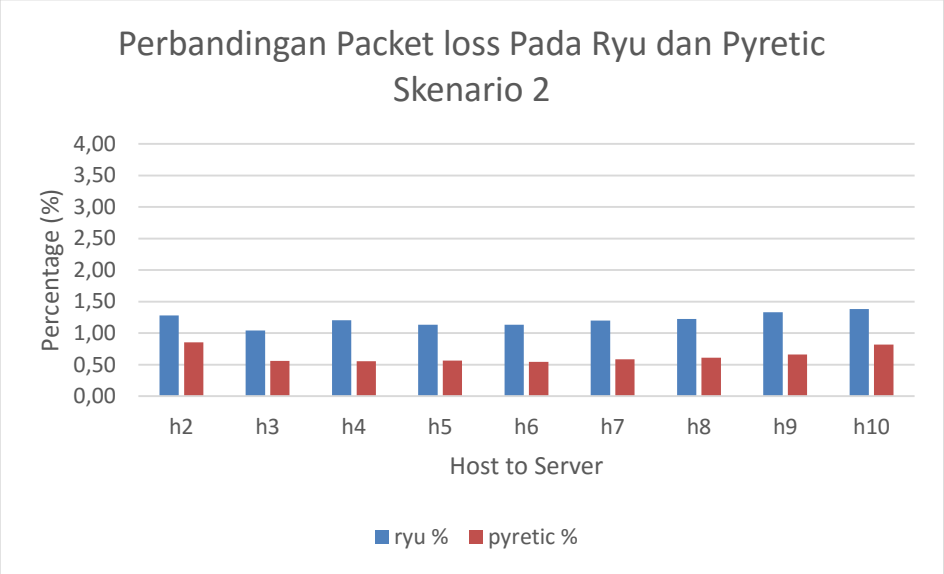
Hasil nilai rata-rata *jitter* skenario 2 sama dengan skenario 1 bahwa secara keseluruhan bahwa nilai *jitter* pada kedua kontroler termasuk sangat bagus dalam kategori degradasi yakni kurang dari 1 ms. Sama halnya dengan skenario 1 nilai rata-rata *jitter* kedua kontroler relatif kecil karena selain *bandwidth* 1024 Mbps, ukuran paket UDP dalam pengujian 1470 *Byte* sehingga waktu *reassemble* paket memiliki waktu yang cepat.

5.1.5.2 Analisis Perbandingan *Packet loss* Pada *Ryu* Dan *Pyretic* Skenario 2

Perbandingan nilai rata-rata *packet loss* seperti pada tabel 5.10 berdasarkan hasil pengujian yang telah dilakukan dengan menggunakan batas *bandwidth* sebasar 100 Mbps. Berikut hasil perbandingan *jitter* pada kedua kontroler seperti pada tabel 5.10.

Tabel 5. 10 Analisis perbandingan *packet loss* pada *Ryu* dan *Pyretic* skenario 2

Packet loss				
Host	**Tujuan**	***Ryu (%)***	***Pyretic (%)***	**Jalur / Path**
2	1	1,280	0,854	2-8-7-3-1
3	1	1,040	0,560	3-1
4	1	1,206	0,556	4-6-1
5	1	1,132	0,564	5-7-3-1
6	1	1,130	0,542	6-1
7	1	1,200	0,586	7-3-1
8	1	1,224	0,608	8-7-3-1
9	1	1,328	0,660	9-5-7-3-1
10	1	1,380	0,820	10-9-5-7-3-1



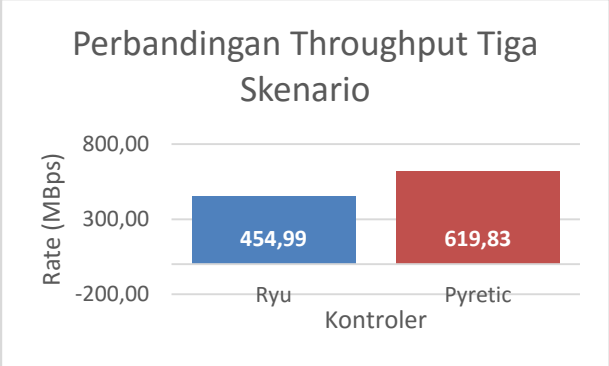
Gambar 5. 10 Perbandingan *packet loss* pada *Ryu* dan *Pyretic skenario 2*

Berdasarkan gambar 5.10 dan tabel 5.10 dapat disimpulkan bahwa secara keseluruhan rata-rata nilai *packet loss* pada skenario 2 mengalami kenaikan yang signifikan. Nilai rata-rata *packet loss* kontroler *Ryu* lebih tinggi dibanding dengan kontroler *Pyretic*. Nilai rata-rata *packet loss* tertinggi terjadi pada host 10 dengan nilai rata-rata *packet loss* 1,38 % untuk *Ryu* dan 0,82 % untuk *Pyretic* pada jalur 10-9-5-7-3-1. Dan terendah pada host 3 dengan nilai rata-rata *packet loss* 1,04 % untuk *Ryu* dan 0,56 % untuk *Pyretic* pada jalur 3-1.

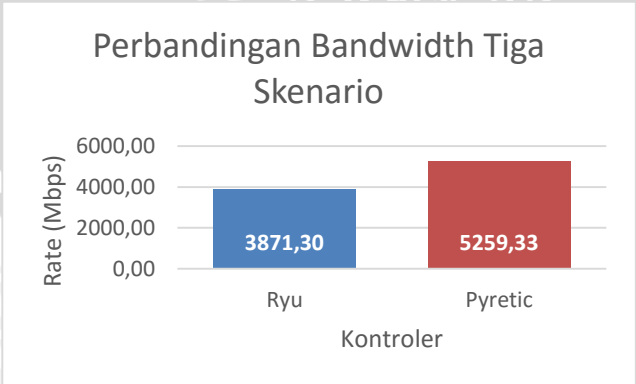
Secara keseluruhan bahwa nilai *packet loss* pada kedua kontroler mengalami tidak terlalu jauh berbeda. Kontroler *Ryu* memiliki nilai rata-rata *packet loss* sedikit lebih besar dan termasuk kategori bagus dalam tabel degradasi dengan nilai rata-rata dibawah 1,5 %. Sedangkan kontroler *Pyretic* memiliki nilai rata-rata *jitter* yang lebih rendah dengan nilai dibawah 1% dan masuk kategori sangat bagus dalam kategori degradasi. Hal ini dipengaruhi dengan banyaknya jumlah data yang ditransfer yang mencapai 823 MB dalam waktu 10 detik karena *bandwidth* yang digunakan lebih besar yakni 1024 Mbps. Selain batas *bandwidth* yang ditentukan yakni 1024 Mbps, ukuran *buffer* UDP dalam pengujian juga sama. Karena jumlah ukuran *buffer* UDP yang sama besar dengan skenario 1 yakni 208 KB dengan jumlah transfer lebih tinggi yakni 823 MB, maka jumlah *packet loss* kedua kontroler akan mengalami kenaikan sepeti pada tabel 5.10 dan gambar 5.10.

5.1.6 Analisis Perbandingan QoS

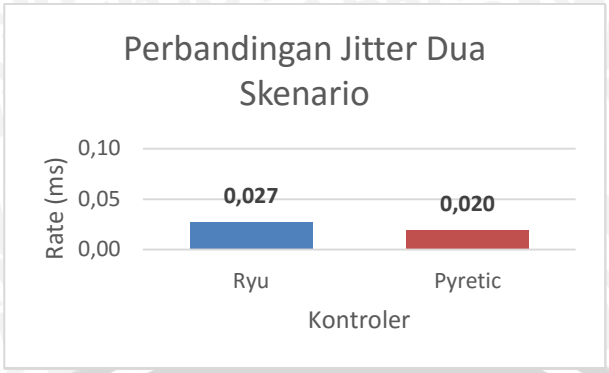
Analisis Perbandingan QoS ini menjelaskan secara ringkas perbandingan rata-rata empat parameter QoS antara kontroler *Ryu* dan kontroler *Pyretic,* yaitu *throughput, bandwidth, jitter,* dan *packet loss*. Nilai perbandingan diambil dari nilai rata-rata seluruh skenario berdasarkan parameter. Berikut adalah grafik perbandingan QoS seperti pada gambar 5.11, 5.12, 5.13, dan 5.14. :



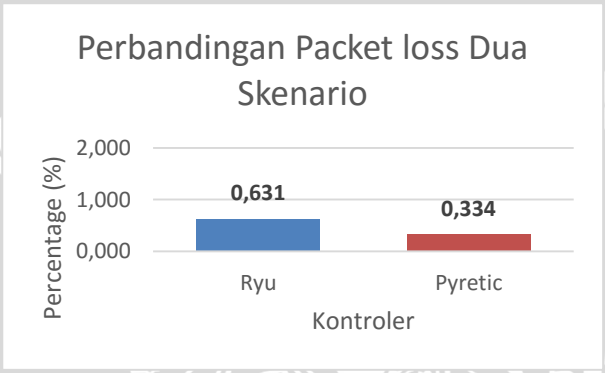
Gambar 5. 11 Perbandingan rata-rata *throughput Ryu* dan *Pyretic* tiga skenario



Gambar 5. 12 Perbandingan rata-rata *bandwidth Ryu* dan *Pyretic* tiga skenario



Gambar 5. 13 Perbandingan rata-rata *jitter Ryu* dan *Pyretic* tiga skenario



Gambar 5. 14 Perbandingan rata-rata *packet loss Ryu* dan *Pyretic* tiga skenario

Perbandingan rata-rata empat parameter yang ditampilkan gambar 5.11, 5.12, 5.13, dan 5.14 menunjukkan perbedaan secara ringkas seluruh pengujian yang telah dilakukan. Hasil perbandingan ini merangkum seluruh perbandingan pengujian pada analisis tiap tahap sebelumnya. Perbandingan tahap 1 yaitu *throughput* dan *bandwidth* ditampilkan menjadi satu dari tiga skenario pengujian. Dan pengujian tahap 2 juga ditampilkan menjadi satu dari dua skenario pengujian.

Nilai rata-rata *throughput Pyretic* mencapai 619,83 MBps lebih besar dibanding kontroler *Ryu* yang hanya 454,99 MBps. Nilai rata-rata *bandwidth Pyretic* mencapai 5259,33 Mbps juga lebih besar dibanding kontroler *Ryu* yang hanya 3871,3 Mbps. Nilai rata-rata *jitter Pyretic* lebih rendah dengan waktu 0,020 ms sedangkan kontroler *Ryu* mempunyai rata-rata waktu 0,027 ms. Dan nilai rata-rata *packet loss Pyretic* juga lebih rendah dengan persentase sebesar 0,334 % sedangkan kontroler *Ryu* mempunyai prosentase sebesar 0,631 %.

Semakin besar nilai rata-rata *throughput* dan *bandwidth* maka semakin baik performa kontroler tersebut dan semakin rendah nilai rata-rata *jitter* dan *packet loss* maka semakin baik pula performa kontroler. Berdasarkan hasil tersebut dapat disimpulkan bahwa kontroler *Pyretic* unggul dibanding kontroler *Ryu* dalam seluruh perbandingan empat parameter QoS. Perbedaan performa pada kontroler tersebut disebabkan karena penggunaan CPU pada kontroler *Ryu* lebih besar dibanding kontroler *Pyretic*. Perbedaaan juga dipengaruhi kontroler *Ryu* yang harus menggunakan protokol *OpenFlow* versi 1.3 sedangkan *Pyretic*

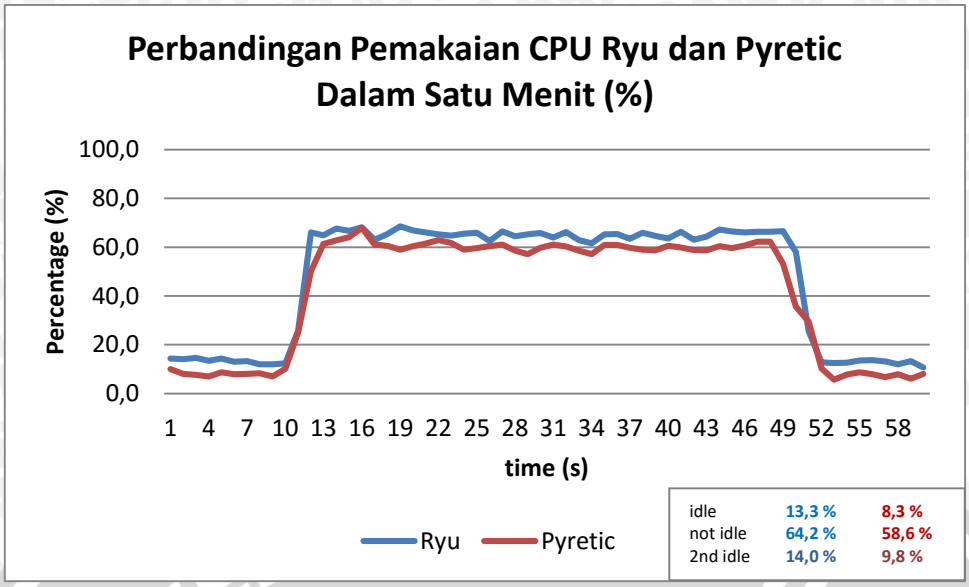
menggunakan *OpenFlow* versi 1.0 ketika menggunakan algortitma *Non-weighted Dijkstra. OpenFlow* versi 1.3 mengalami perbaikan IPv6, QoS, dan dukungan *Q-in-Q Tunneling*. Perbedaan performa kedua kontroler tidak dipengaruhi beban RAM yang digunakan. Beban RAM yang digunakan pada kedua kontroler hampir sama yaitu mencapai 13,6 % pada kontroler Ryu dan 12,7 % pada kontroler *Pyretic*.

5.1.7 Analisis Perbandingan Pemakaian CPU dan RAM Pada *Ryu* dan *Pyretic*

Analisis terhadap pemakaian CPU dan RAM pada kedua kontroler ini berdasarkan hasil pengujian sebelumnya. Perbandingan dilakukan pada kedua kontroler untuk mengetahui perbedaan pemakaian CPU dan RAM dengan menggunakan lama waktu 60 detik. Data hasil pengujian kedua kontroler dibandingkan dengan menggunakan tabel seperti pada tabel 5.11 dan gambar 5.15 berikut:

Tabel 5. 11 Perbandingan Rata-rata Pemakaian CPU *Ryu* dan *Pyretic* Pada *Host* 2

Detik ke	10 detik pertama *idle* (%)		10 detik kedua (%)		10 detik ketiga (%)		10 detik keempat (%)		10 detik kelima (%)		10 detik keenam *idle* (%)	
	Ryu	*Pyretic*	*Ryu*	*Pyretic*	*Ryu*	*Pyretic*	*Ryu*	*Pyretic*	*Ryu*	*Pyretic*	*Ryu*	*Pyretic*
1	14,3	10,0	25,8	25,1	66,0	61,4	63,9	61,1	66,3	59,9	25,5	29,4
2	14,1	8,0	66,0	50,2	65,3	62,9	66,1	60,3	63,1	58,8	12,7	10,3
3	14,6	7,6	64,9	61,3	64,8	61,8	62,9	58,6	64,4	58,8	12,5	5,6
4	13,4	7,0	67,7	62,8	65,6	59,0	61,6	57,2	67,3	60,4	12,7	7,7
5	14,3	8,6	66,7	64,0	66,0	59,6	65,3	60,9	66,5	59,7	13,5	8,6
6	13,1	7,8	68,2	68,0	62,5	60,4	65,4	60,9	66,0	60,6	13,6	7,9
7	13,3	8,1	63,1	61,1	66,5	61,1	63,4	59,8	66,3	62,2	13,2	6,7
8	12,0	8,3	65,4	60,5	64,5	58,6	65,9	59,0	66,3	62,3	12,0	7,9
9	12,0	7,0	68,6	58,9	65,3	57,1	64,6	58,7	66,6	53,3	13,3	6,0
10	12,4	10,1	66,8	60,4	65,8	59,7	63,6	60,6	58,1	35,6	10,7	8,0

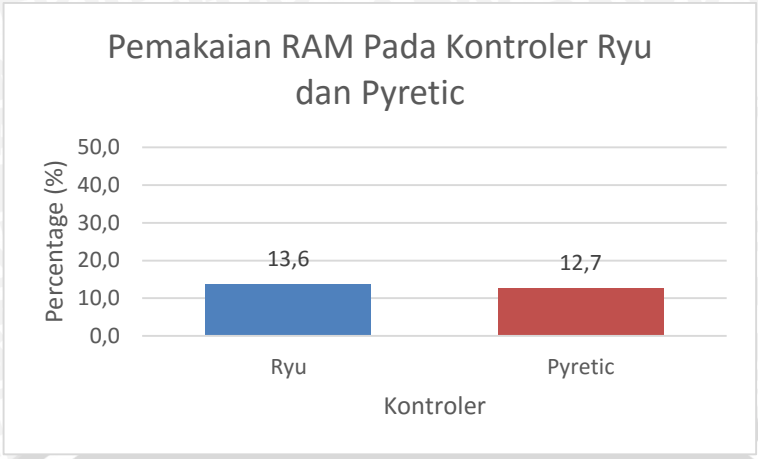


Gambar 5. 15 Perbandingan Pemakaian CPU *Ryu* dan *Pyretic* Pada *Host* 2

Berdasarkan tabel 5.11 dan gambar 5.16 dapat diketahui bahwa pengamatan ini dilakukan selama 60 detik dengan membandingkan pemakaian CPU pada kontroler *Ryu* dan *Pyretic*. Pengujian ini adalah hasil rata-rata pemakaian CPU yaang digunakan pada kontroler *Ryu* dan *Pyretic.* Pengujian dilakukan terhadap pemakaian CPU saat pengujian QoS pada tahap 1 dan tahap 2.

Hasil pengamatan menunjukkan bahwa selama 10 detik pertama yaitu ketika kontroler berjalan namun belum melakukan pengujian apapun atau *idle* mempunyai rata-rata pemakaian CPU pada kontroler *Ryu* sebesar 13,3 % dan 8,3 % pada kontroler *Pyretic*. Hasil pengamatan pemakaian CPU pada 40 detik selanjutnya menunjukkan bahwa pemakaian CPU pada kontroler *Ryu* mencapai rata-rata 64,2 % lebih rendah daripada kontroler *Pyretic* yang rata-ratanya mencapai 58,6 %. Hal ini berarti penggunaan kontroler *Ryu* membutuhkan *resourse hardware* yang lebih tinggi. Hasil pengamatan pemakaian CPU pada 10 detik terakhir menunjukkan perubahan pemakaian CPU setelah pengujian QoS dilakukan. Hasilnya menujukkan bahwa rata-rata pemakaian CPU pada kedua kontroler *Ryu* mengalami penurunan. Pemakaian CPU pada kontroler *Ryu* menunjukkan angka rata-rata sebesar 14 % sedangkan pemakaian CPU pada kontroler *Pyretic* menunjukkan angka rata-rata sebesar 9,8 %, sedangkan nilai prosentase CPU tertinggi pada saat pengujian mencapai 68,6 % pada kontroler *Ryu* dan 68 % pada kontroler *Pyretic*.

Secara keseluruhan pemakaian CPU pada kontroler *Ryu* lebih tinggi dibanding dengan pemakaian CPU pada kontroler *Pyretic*. Persentase pemakaian CPU pada kontroler *Pyretic* lebih rendah baik dalam keadaan *idle* maupun pengujian QoS yang berarti Kontroler *Ryu* membutuhkan spesifikasi hardware yang lebih tinggi dibanding dengan *Pyretic.*



Gambar 5. 16 Pemakaian RAM pada kontroler *Ryu* dan *Pyretic*

Berdasarkan gambar 5.17 dapat diketahui bahwa pengamatan ini dilakukan dengan membandingkan pemakaian RAM pada kontroler *Ryu* dan *Pyretic*. Pengujian dilakukan terhadap pemakaian CPU saat pengujian QoS pada tahap 1 dan tahap 2. Pengujian pemakaian RAM dilakukan bersamaan dengan pengujian pemakaian CPU. Hasil pengujian ini merupakan data tunggal rata-rata dari lima kali pengujian. Data yang ditampilkan pada gambar merupakan persentase penggunaan RAM dari jumlah RAM total 2650 MB.

Secara keseluruhan pemakaian RAM pada kedua kontroler relatif rendah. Namun kontroler *Ryu* memiliki jumlah pemakaian lebih besar dibanding dengan pemakaian pada kontroler *Pyretic*. Hasil perbandingan menunjukkan bahwa rata-rata pemakaian RAM kontroler *Ryu* mencapai 13,6 % sedangkan kontroler *Pyretic* mencapai 12,7 %. Pemakaian RAM ini merupakan jumlah beban yang diterima sistem ketika sedang menjalankan pengujian terhadap QoS. Aplikasi yang dijalankan saat pengujian adalah Mininet, Iperf, dan Kontroler. Aplikasi tersebut tidak termasuk aplikasi sistem dari sistem operasi yang sedang berjalan.

BAB 6 KESIMPULAN

Bab ini berisi kesimpulan dari hasil pengujian dan analisis sistem yang telah dilakukan pada bab selanjutnya serta saran untuk pemilihan kontroler pada arsitektur SDN.

6.1 Kesimpulan

Dari hasil pengujian dan analisis performa kontroler *Ryu* dan *Pyretic* pada parameter *throughput, bandwidth, jitter,* dan *packet loss,* dapat disimpulkan bahwa :

1. Arsitektur jaringan SDN diterapkan dengan menggunakan topologi yang bisa disimulasikan implementasikan Mininet. Topologi dibuat dengan satu kontroler yang menghubungkan banyak 10 *switch*. Setiap switch terhubung dengan *host.* Pengelolaan jaringan dilakukan tanpa harus menyentuh *switch* melainkan hanya melalui kontroler. Salah satu kontroler *Ryu* atau *Pyretic* dapat dijalankan setelah topologi dijalankan terlebih dahulu. Penggunaan Kontroler dilakukan secara bergantian.
2. Analisis *throughput, bandwidth, jitter, packet loss* menggunakan Iperf dalam jaringan SDN dapat dilakukan dengan menggunakan simulasi pengujian pada Mininet. Pengujian dalam penelitian ini menerapkan aplikasi *Non-weighted Dijkstra.* Pengujian dilakukan dengan membuat *host* 1 sebagai *server* dan *host* 2 sampai *host* 10 sebagai *client*. Iperf digunakan dengan mengatur opsi perintah dalam terminal . Pengujian parameter QoS dilakukan secara bergantian ditiap *client* terhadap *server* selama 10 detik. Pengujian pemakaian CPU dilakukan dengan cara merekam aktivitas CPU ketika keadaan *idle* dan ketika sedang menguji *throuhput* selama 60 detik. Pengujian performa *throughput, bandwidth, jitter, packet loss* dan pemakaian CPU dilakukan dengan menggunakan dua kontroler secara bergantian. Selanjutnya data hasil pengujian *throughput, bandwidth, jitter, packet loss* dan pemakaian CPU dapat dianalisis.
3. Berdasarkan analisis yang telah dilakukan rata-rata nilai *throughput* pada kontroler *Pyretic* lebih tinggi dibanding dengan nilai *throuhput* pada kontroler *Ryu*. Berdasarkan rata-rata tiga skenario pengujian, nilai rata-rata *throughput* pada kontroler *Pyretic* sebesar 619,83 MBps, sedangkan kontroler *Ryu* memiliki nilai rata-rata *throughput* sebesar 454,99 MBps. Tinggi rata-rata *throughput* dipengaruhi oleh *TCP window size* yang digunakan dalam pengujian. Selain itu nilai rata-rata *throughput* juga dipengaruhi pada jalur transmisi yang dilalui. Jika jalur semakin banyak *switch* yang dilalui maka nilai *throughput* akan semakin rendah. Perbedaan performa kontroler terjadi karena perbedaan versi protokol *OpenFLow.*
4. Hasil analisis *bandwidth* menunjukkan bahwa kontroler *Pyretic* juga memiliki nilai rata-rata *bandwidth* yang lebih besar dibanding dengan kontroler *Ryu.* Berdasarkan rata-rata *bandwidth* tiga skenario pengujian,

nilai rata-rata *bandwidth* pada *Pyretic* mencapai 5259,33 Mbps atau 5,13 Gbps dan 3871,3 Mbps atau 3,78 Gbps pada *Ryu*. Hampir sama dengan *throughput*, tinggi rata-rata *bandwidth* dipengaruhi oleh *TCP window size* yang digunakan dalam pengujian dan jalur transmisi yang dilalui. Jika jalur semakin banyak *switch* yang dilalui maka nilai *throughput* akan semakin rendah.

5. Berdasarkan analisis terhadap *jitter* dapat disimpulkan bahwa secara keseluruhan rata-rata nilai *jitter* pada kontroler *Pyretic* lebih rendah dibanding dengan kontroler *Ryu*. Nilai rata-rata *jitter* pada kontroler *Pyretic* mencapai 0,020 ms sedangkan kontroler *Ryu* mencapai 0,027 ms. Nilai tersebut masih terbilang sangat bagus dalam kategori degradasi. Hasil analisis juga menunjukkan bahwa nilai *jitter* kedua kontroler relatif stabil dalam keseluruhan pengujian walaupun kontroler *Ryu* tidak lebih stabil daripada kontroler *Pyretic*. Nilai rata-rata *jitter* yang relatif kecil juga dipengaruhi oleh ukuran paket UDP yang relatif kecil sebesar 1470 *Byte*.
6. Secara keseluruhan bahwa nilai rata-rata *packet loss* pada kontroler *Pyretic* lebih rendah daripada nilai *packet loss* pada *Ryu*. Nilai rata-rata *paket loss Pyretic* mencapai 0,334 % sedangkan *packet loss* pada *Ryu* mencapai 0,631 %. Perbedaan anatara kedua kontroler tidak terlalu jauh. Tingkat *packet loss* dipengaruhi oleh *bandwidth* yang diujikan dan ukuran *buffer* UDP yang sebesar 208 KB. Berdasarkan rata-rata *packet loss* seluruh skenario, kedua kontroler masih memiliki rata-rata *packet loss* yang sangat baik menurut tabel degradasi.
7. Hasil analisis rata-rata pemakaian CPU menunjukkan bahwa prosentase pemakaian CPU kontroler *Pyretic* lebih sedikit dibanding kontroler *Ryu*. Pada fase *idle Pyretic* membutuhkan rata-rata 8,3 % CPU, sedangkan *Ryu* membutuhkan rata-rata 13,3 %. Ketika dilakukan pengujian QoS, *Pyretic* membutuhkan rata-rata 58,6 % CPU sedangkan *Ryu* membutuhkan rata-rata 64,2 % CPU. Sedangkan hasil perbandingan pemakaian RAM menunjukkan bahwa rata-rata pemakaian RAM kontroler *Ryu* mencapai 13,6 % sedangkan kontroler *Pyretic* mencapai 12,7 %. Hal ini menunjukkan bahwa kontroler *Ryu* membutuhkan perangkat *hardware* dengan spesifikasi yang lebih tinggi dibanding kontroler *Pyretic*.

6.2 Saran

Saran yang dapat penulis sampaikan untuk penelitian lebih lanjut terkait dengan penelitian ini adalah:

1. Diperlukan pengujian lebih lanjut tentang analisis performa kontroler yang menggunakan *physical switch* dan *physical host* agar hasilnya dapat dibandingkan dengan penelitian menggunakan simulator.
2. Diperlukan pengujian dengan menggunakan parameter lainnya. Parameter yang dapat digunakan adalah delay atau parameter lain.

3. Penelitian selanjutnya dapat menggunakan aplikasi atau algoritma lain yang lebih bervariasi misal *extended dijkstra algoritm*.
4. Skenario pengujian dapat dikembangkan dengan menggunakan skenario yang lebih bervariasi.
5. Penelitian selanjutnya dapat menggunakan kontroler turunan ataupun kontroler lainnya.
6. Penelitian selanjutnya dapat menggunakan topologi yang lebih bervariasi dan kompleks atau lebih dari satu topologi

DAFTAR PUSTAKA

Al-Somaidai, M. B. & Yahya, E. B., 2014. Survey of software components to emulate *OpenFlow*. *American Journal of Software Engineering and Applications,* Volume 3, pp. 74-78.

Anggara, S. M., 2015. *Pengujian Performa Kontroler Software-defined Network (SDN): POX dan Floodlight.* Bandung: Institut Teknologi Bandung.

Ashton, M., 2014. *Ten Things to Look For in an SDN Controller.*

Chaudhary, D. D., & Waghmare, L. M., 2012. *Quality of service analysis in wireless sensor network by controlling end-to-end delay*. pp. 703-708. IEEE.

ETSI, 1999. *Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); General aspects of Quality of Service (QoS).* V2.1.1 penyunt. Valbonne: European Telecommunications Standards Institute.

Hu, F., Hao, Q. & Bao, K., 2014. *A Survey on Software-defined Network and OpenFlow: From Concept to Implementation.* Alabama: University of Alabama.

Iperf., *iPerf - The ultimate speed test tool for TCP, UDP and SCTP.* [online] tersedia di : < https://iperf.fr/iperf-doc.php> [diakses pada 8 April 2016]

Jiang, J. R., Huang, H. W. & Chen, S. Y., 2014. *Extending Dijkstra's Shortest Path Algorithm for Software Defined Network.* Taiwan: National Central University.

Kim, H. & Feamster, N., 2013. *Improving Network Management with Software Defined Networking.* Georgia: Georgia Institue of Technology.

Mendonca, M. et al., 2013. *A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks.*

Pratama, P. N., 2015. *ANALISIS PERFORMANSI KONTROLER FLOODLIGHT DAN BEACON SEBAGAI KOMPONEN UTAMA PADA ARSITEKTUR SOFTWARE DEFINED NETWORK.* Malang: Universitas Brawijaya.

Rao, S., 2015. *SDN Series Part Eight: Comparison Of Open Source SDN Controllers.*The New Stack.

Reich, J. et al., 2013. *Modular SDN Programming with Pyretic.*

Ross, K. & Kurose, J., 2013. *Computer Networking : a top-down approach.* 6 penyunt. New Jersey: Pearson Education, Inc.

Seehorn, A., 2016. *What is the Difference Between CPU Usage & RAM?*. [online] tersedia di : <https://www.techwalla.com/articles/what-is-the-difference-between-cpu-usage-ram> [diakses pada 28 November 2016]

Silver, J., 2013. *SDN 101: What It Is, Why It Matters, and How to Do It.* [online] tersedia di : <http://blogs.cisco.com/ciscoit/sdn-101-what-it-is-why-it-matters-and-how-to-do-it> [diakses pada 8 April 2016]

Underdahl, B. & Kinghorn, G., 2015. *Software Defined Networking For Dummies.* New Jersey: John Wiley & Sons, Inc.

Wang, S. Y., Chiu, H. W. & Chou, C. L., 2015. *Comparisons of SDN OpenFlow Controllers over EstiNet: Ryu vs. NOX.* Taiwan: National Chiao Tung University.

Wibowo, M., 2014. *Analisis dan Implementasi Quality Of Service (Qos) Menggunakan Ipcop Di Smk Muhammadiyah Imogiri*. Yogyakarta: AMIKOM YOGYAKARTA