ANALISIS PERBANDINGAN KINERJA METODE *LOAD BALANCING* BERBASIS *CPU* DENGAN BERBASIS *RESPONSE TIME* PADA *SOFTWARE DEFINED NETWORK*

SKRIPSI

Untuk memenuhi sebagian persyaratan
Memperoleh gelar Sarjana Komputer

Disusun oleh:
Muharrom Abdillah
NIM: 135150201111261

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

ANALISIS PERBANDINGAN KINERJA METODE *LOAD BALANCING* BERBASIS *CPU* DENGAN BERBASIS *RESPONSE TIME* PADA *SOFTWARE DEFINED NETWORK*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Muharrom Abdillah
NIM: 135150201111261

Skripsi ini telah diuji dan dinyatakan lulus pada
02 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Widhi Yahya, S.Kom., M.Sc.	Achmad Basuki, S.T, M.MG, Ph.D
NIK: 2016078911211001	NIP: 19741118 200312 1002

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, 2 Agustus 2018

Muharrom Abdillah

NIM: 135150201111261

KATA PENGANTAR

Segala puji dan syukur penulis ucapkan atas kehadirat Allah SWT yang telah memberikan Karunia dan Rahmat-Nya sehingga penulis mampu menyelesaikan penelitian ini dengan judul "Analisis Perbandingan Kinerja Metode *Load Balancing Berbasis CPU dengan Metode Load Balancing Berbasis Response Time Pada Software Defined Network*". Penelitian ini dilakukan untuk memenuhi syarat untuk mendapatkan gelar sarjana di Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya. Selain itu penelitian ini juga dapat terselesaikan atas bantuan, dukungan serta bimbingan dari berbagai pihak. Untuk itu penulis ingin mengucapkan banyak terima kasih kepada:

1. Yang terhormat bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya.
2. Yang terhormat bapak Tri Astoto Kurniawan , S.T.,M. T,Ph. D selaku ketua jurusan Teknik Informatika.
3. Yang terhormat bapak Widhi Yahya S.Kom., M.Sc sebagai dosen pembimbing 1 yang telah banyak meluangkan waktunya untuk memberikan ilmu dan saran kepada penulis terkait pengerjaan penelitian ini.
4. Yang terhormat bapak Achmad Basuki, S.T, M.MG, Ph.D sebagai dosen pembimbing 2 yang sangat sabar memberikan arahan dan bimbingan kepada penulis dalam proses penelitian.
5. Seluruh dosen Fakultas Ilmu Komputer Universitas Brawijaya yang telah mengajarkan ilmu pengetahuan dan memberikan motivasi selama menjalani perkuliahan.
6. Orang tua penulis yang selalu mempertanyakan proses pengerjaan skripsi ini sehingga penulis selalu sadar untuk menyelesaikan penelitian ini.
7. Istri tercinta yang setiap saat selalu memberi dukungan dan do'a dalam pengerjaan skripsi ini, sampai akhirnya skripsi ini dapat terselesaikan.
8. Teman-teman khususnya Muhammad Sholeh, Riski Julianto dan Hasbi Razzak, yang selalu memberikan bantuan, saran dan motivasi kepada penulis dalam pengerjaan skripsi.
9. Teman-teman seperjuangan lain yang tidak dapat penulis sebutkan satu persatu disini atas seluruh bantuan, saran dan motivasi yang telah diberikan.

Terakhir, penulis sangat berharap semoga seluruh kebaikan mereka mendapatkan balasan dari Allah SWT. Penulis menyadari bahwa skripsi ini jauh dari kesempurnaan, oleh karena itu penulis mengharapkan saran dan kritik untuk pengembangan skripsi ini selanjutnya agar mencapai hasil yang lebih baik dan

dapat berkontribusi memberikan pengetahuan khususnya pada bidang Ilmu Komputer. Semoga penelitian yang dikerjakan oleh penulis ini dapat bermanfaat bagi pembaca khususnya teman-teman mahasiswa Fakultas Ilmu Komputer Universitas Brawijaya.

Malang, 2 Agustus 2018

Penulis

muharromab@gmail.com

ABSTRAK

Server merupakan bagian penting dalam sebuah layanan didalam jaringan komputer. Peran *server* dapat menentukan kualitas baik buruknya dari layanan tersebut. Semakin banyak *request*, akan membuat *server* terbebani dan membuat kinerja dari *server* menjadi buruk. Solusinya adalah dengan menggunakan *multiple server* yang menerapkan metode *load balancing* agar kinerja *server* tetap stabil. Metode *Load balancing* diimplementasikan pada *software defined network.* Arsitektur ini menawarkan *scalability* dan *programmability* untuk penggunaan jaringan yang semakin kompleks seperti *load balancing web server.*

Ada beberapa penelitian tentang *load balancing* pada *SDN* dengan berbagai metode yang diterapkan diantaranya yaitu metode *round robin,*metode berbasis *CPU usage* dan metode berbasis *response time*. Tentunya, kinerja yang dihasilkan dari metode yang digunakan akan berbeda-beda. Penelitian ini memberikan analisis terhadap kinerja dari metode *round robin*, metode berbasis *CPU* dan metode berbasis *response time*. Pengujian dilakukan dengan tiga kategori *rate* yaitu *low* untuk *rate 40, medium* untuk *rate 80* dan *high* untuk *rate 160*. Pengujian dari masing-masing *rate* dilakukan dalam waktu 10 detik. Parameter pengujian yang digunakan adalah *throughput*, *response time*, dan *CPU Usage*.

Hasil pengujian *throughput* menunjukkan bahwa metode *round robin* unggul pada pengujian kategori *low* dengan nilai rata-rata 32,64 KB/s, sedangkan pada pengujian kategori *medium* dan *high* metode berbasis *response time* masih lebih unggul dengan nilai rata-rata 65,02 KB/s dan 115,50 KB/s . Pada pengujian *response time* menunjukkan metode berbasis *response time* memiliki *response time* paling cepat di semua kategori *rate* dengan nilai rata-rata 30,32 ms, 67,56 ms, dan 118,48 ms. Sementara pada pengujian *CPU usage,* metode berbasis *CPU* memiliki standar deviasi yang terendah dari semua kategori *rate* dengan nilai rata-rata 3,95%, 4,69% dan 5,90%.

Kata kunci : *load balancing, round robin, CPU, response time, software defined network.*

ABSTRACT

Server is part of important thing in a service inside computer network. the purpose of server can determine how fine the quality of the services. More request will make the server have more loads and decreasing performance of server. The solution is to use multiple servers that implement load balancing method, so that servers performance becomes more stable. Load balancing method implemented in software defined network. This architecture offers scalability and programmability for more complex network like load balancing for web server

There are any research about load balancing in SDN with several method implemented, they are round robin method, CPU usage based method and response time based method. Certainly, the result of performance for each method are different. This research give analysis to performance of round robin method, CPU usage based method and response time based method. The trials are doing to three categories of rates, they are low for rate 40, medium for rate 80 and high for rate 160. The trials are doing in 10 seconds for each rate. The parameters used in this trial are throughput, response time and cpu usage.

The throughput test resulting that round robin is excellent in low category with average value 32,64 KB/s, meanwhile in medium and high category response time method is more excellent with average values 65,02 KB/s and 115,50 KB/s. On response time test, showing that response time method have the fastest response time than all rate category with average values 30,32 ms, 67,56 ms and 118,48 ms. Whereas, in CPU usage test, CPU usage method have the lowest standart deviation with average values 3,95%, 4,69%, and 5,90%.

Keywords: *load balancing, round robin, CPU, response time, software defined network*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Server merupakan bagian penting dalam sebuah layanan didalam jaringan komputer. Peran *server* dapat menentukan kualitas baik buruknya dari layanan tersebut. Jika *server* yang dimiliki hanya ada satu dan bekerja tunggal, maka memungkinkan terjadinya "*a single point of failure*" *(SPOF)*. *SPOF* adalah kondisi *server* yang gagal memberi respon dan mengakibatkan sistem tidak berfungsi atau mati (Moniruzzaman, 2015). Masalah ini terjadi karena terlalu banyak *request* yang harus di *handle* oleh satu *server*. Solusi untuk mengatasi masalah tersebut adalah dengan menggunakan *multiple server*. Didalam *multiple server* dibutuhkan metode dalam pembagian beban agar kinerja *server* tetap stabil. Metode yang digunakan adalah metode *load balancing*.

Load balancing adalah teknik untuk mendistribusikan beban pada dua atau lebih jalur koneksi secara seimbang agar dapat berjalan optimal, memaksimalkan *throughput*, memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi (Arianto & Sholeh, 2014). *Load balancing* akan membagikan beban yang datang ke beberapa *server*, gunanya untuk menghindari *overload* pada salah satu jalur koneksi. Umumnya metode *load balancing* sendiri sudah banyak diimplementasikan pada perangkat internet sekarang. Kekurangannya, pada perangkat yang sering digunakan sekarang *control plane* dan *data plane* tertanam pada perangkat jaringan yang sama, dalam artian *control plane* dan *data plane* tidak dipisah. Jadi apabila ingin bereksperimen dengan beberapa teknik seperti teknik *load balancing* baru, maka tidak akan dapat dilakukan. Karena pada perangkat internet sekarang, sistem operasi dan fitur-fiturnya sudah langsung *embedded* pada perangkat tersebut. Oleh karena itu kita memerlukan teknologi yang mampu menangani penggunaan jaringan yang sangat kompleks yaitu *Software defined network* (*SDN*).

SDN merupakan sebuah cara baru untuk mengelola, mendesain dan mengimplementasikan jaringan dimana *control plane* dan *data plane* dibuat terpisah. Arsitektur *SDN* dapat memudahkan dalam melakukan pemrograman secara langsung. Agar dapat berkomunikasi antara perangkat dan *controller* pada *SDN* menggunakan sebuah *protocol* yang disebut dengan *openflow*. *Openflow* adalah *protocol* untuk mengontrol *data plane* yang secara fisik sudah terpisah dari *control plane* menggunakan perangkat lunak yaitu *controller* pada sebuah *server* (Kartadie & Utami, 2014). Dengan adanya konsep *SDN* operator atau *network administrator* sangat dimudahkan dalam melakukan pengelolahan jaringan. Karena konsep utama pada arsitektur ini adalah sentralisasi kendali jaringan dengan semua pengaturan berada pada *control plane* (Ummah & Abdillah, 2016).

Penelitian tentang *load balancing* pada *SDN* sudah pernah dilakukan sebelumnya dengan berbagai metode dan menghasilkan kinerja yang berbeda-beda. Pertama "Implementasi *Load Balancing* di *Web Server* Menggunakan

Metode Berbasis Sumber Daya CPU Pada *SDN"* (Julianto, 2017). Penelitian ini mencoba mengembangkan metode *load balancing* dengan mempertimbangkan kondisi dari *server* dengan cara mengambil sumber daya *CPU* yang paling ringan atau kecil. Pengujian pada penelitian ini dilakukan dengan menggunakan *server heterogen* atau dengan kapasitas *server* yang berbeda. Hasil yang didapatkan metode berbasis *CPU usage* dapat berjalan dengan baik dengan membagi beban paling banyak pada *server* yang memiliki jumlah *core* paling besar dibandingkan pada *server* yang memiliki jumlah *core* kecil. Penelitian yang ke dua adalah "An Efficient SDN Load balancing Scheme Based on *Server response time*" (Zhong & Fang, 2016). *SDN controller* akan mengambil sampel *server response time* dari *server* dengan mengirimkan pesan *paket_out* kepada *server*. Kemudian *server* membalas dengan mengirimkan *paket_in* dan dilakukan perhitungan. Hasilnya akan dibandingkan, kemudian ketika client melakukan request akan diarahkan pada server dengan response time paling kecil. Penelitian ini dilakukan pada *server homogen* atau dengan kapasitas *server* yang sama. Hasil yang diperoleh metode berbasis *response time* dapat berjalan dengan baik dengan mendistribusikan beban pada *server* yang memiliki *response time* tercepat. Pengujian pada penelitan ini dilakukan dengan menggunakan *server homogen.*

Dari penelitian yang sudah diakukan sebelumnya, maka peneliti berfokus pada analisis perbandingan kinerja yaitu "Analisis Perbandingan Kinerja Metode Berbasis *CPU* dan Metode berbasis *response time* untuk *Load balancing* pada *Software Defined Network*". Penelitian ini ditujukan untuk mencari kinerja terbaik dengan menggunakan parameter yang sudah ditentukan dari metode berbasis *CPU* dan metode berbasis *response time*. Pada penelitian ini juga di lakukan analisis terhadap metode *round robin* yang digunakan sebagai pembanding terhadapa kedua metode tersebut. Alasan peneliti membandingkan metode berbasis *CPU* dan metode berbasis *response time* adalah karena pada metode pada metode berbasis *CPU* dan metode berbasis *response time* proses *load balancing* sama-sama dilakukan dengan melihat kondisi dari *server*. Selain itu, terdapat korelasi antara metode berbasis *CPU* dengan metode berbasis *response time*, dimana kondisi *CPU* dari *server* akan mempengaruhi waktu respon dari *server*. Sedangkan, alasan metode *round robin* digunakan sebagai pembanding adalah karena pada metode *round robin* proses *load balancing* dilakukan tanpa melihat kondisi dari *server.*

1.2 Rumusan Masalah

Dari latar belakang yang telah dipaparkan, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana cara mengimplementasikan metode *round robin*, metode berbasis *CPU usage*, dan metode berbasis *response time* untuk *load balancing* pada *SDN*?
2. Bagaimana analisis perbandingan metode *round robin,* metode berbasis *CPU usage* dan metode berbasis *response time* pada *server* heterogen untuk *load balancing* pada *SDN*?

1.3 Tujuan

1. Untuk mengetahui bagaimana cara mengimplementasikan metode berbasis *CPU usage*, metode *round robin* dan metode berbasis *response time* untuk *load balancing* yang diterapkan pada arsitektur *SDN?*
2. Mengetahui perbandingan kinerja terhadap metode-metode tersebut berdasarkan parameter *throughput*, *response time* dan *CPU usage*.

1.4 Manfaat

Manfaat yang bisa diperoleh dari penelitian ini diantaranya yaitu:

1. Dapat mengimplementasikan *load balancing* pada *software defined network* menggunakan metode *berbasis CPU usage*, metode *round robin* dan metode berbasis *response time*.
2. Dapat mengetahui hasil uji dari perbandingan antara metode berbasis *CPU usage*, metode *round robin* dan metode berbasis *response time*.
3. Dapat digunakan sebagai acuan dalam penggunaan metode yang efisien untuk *load balancing*.

1.5 Batasan Masalah

Agar penelitian dapat mencapai sasaran dan tujuan yang diharapkan, maka permasalahan yang dibatasi sebagai berikut:

1. Penelitian dilakukan dengan menggunakan tiga buah *web server,* satu buah *SDN switch* dan satu buah *SDN controller.*
2. *SDN switch* yang diterapkan pada sebuah *PC*.
3. Pada sistem *load balancing* hanya melayani trafik *HTTP.*
4. Data yang digunakan untuk analisis merupakan data yang didapatkan dari hasil implementasi oleh lingkungan kerja peneliti.

1.6 Sistematika Pembahasan

Sistem penulisan dalam peneltian ini dibagi menjadi beberapa bab, yaitu:

BAB 1 PENDAHULUAN

Pada bagian ini akan dibahas latar belakang yaitu bagaimana masalah itu ada serta solusi yang bisa dicapai, rumusan masalah yaitu pertanyaan – pertanyaan mengenai hal apa yang diteliti, tujuan penelitian, manfaat penelitian yang diinginkan bagi penulis serta bagi masyarakat, serta batasan masalah.

BAB 2 LANDASAN KEPUSTAKAAN

Menjelaskan kajian pustaka dan dasar teori yang mendasari proses analisis kinerja metode *Berbasis CPU usage, round robin* dan *Berbasis response time* untuk *load balancing* pada *software defined network.*

BAB 3 METODOLOGI

Menjelaskan beberapa tahapan dalam penelitian meliputi: studi literasi, analisis kebutuhan, perancangan sistem, implementasi sistem, pengujian dan analisis dan cara menarik kesimpulan serta saran.

BAB 4 HASIL

Membahas hasil dari pengujian perangkat dan pengujian sistem sesuai dengan perancangan *load balancing* dengan metode *berbasis CPU usage,* metode *round robin* dan metode berbasis *reponse time* pada *software defined network*.

BAB 5 PEMBAHASAN

Menjelaskan dari hasil berupa grafik dan dijelaskan perbandingan kinerja metode berbasis *CPU,* metode *round robin* dan metode berbasis *response time* untuk *load balancing* pada *software defined network* sesuai dengan perancangan sistem yang sudah dibuat.

BAB 6 PENUTUP

Menjelaskan kesimpulan yang diperoleh dari hasil implementasi dan pengujian serta saran untuk pengembangan sistem lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

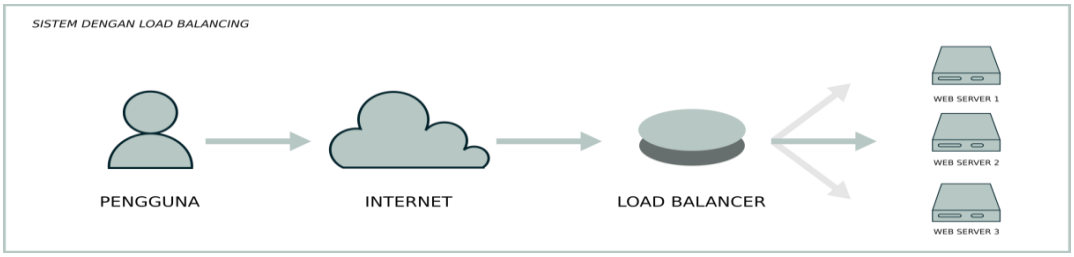
Kajian pustaka bertujuan sebagai pedoman dalam pelaksanaan penelitian. Kajian pustaka diambil dari beberapa penelitian yang pernah dilakukan sebelumnya. Seperti penelitian yang dilakukan sebelumnya yaitu "Implementasi *Load balancing* di *Web Server* dengan metode *round robin* pada *SDN*" (Mahdiyanah, 2016). Tujuan dari penelitian ini agar web *server* dapat membagi beban secara optimal dengan menggunakan metode *round robin* pada *Software difined network.* Kesimpulan dari penelitian ini metode *round robin* dapat berjalan baik dengan membagi beban secara bergantian.

Penelitian selanjutnya, "Implementasi *Load Balancing* di Web *Server* Menggunakan *Metode Berbasis Sumber Daya CPU* Pada *SDN"* (Julianto, 2017). Tujuan dari penelitian ini adalah menggunakan metode *resource CPU di load balancing* pada *SDN*, dengan mengetahui *resource CPU* disetiap *server* diharapkan dapat membuat pembagian beban lebih stabil dan memperkecil kemungkinan *server* mengalami *overload*. Kesimpulan yang didapat dari penelitian ini, metode *resource CPU* di *load balancing* dapat berjalan baik dan dengan membagi beban pada *server* yang memiliki nilai *resource CPU* terkecil.

Penelitian lain yang terkait dengan sistem *load balancing* adalah "An Efficient SDN Load Balancing Scheme Based on *Server response time*" (Zhong & Fang, 2016). Penelitan ini menerapkan metode berbasis *response time* di load balancing pada *SDN.* Hasil simulasi dengan *mininet* menunjukkan bahwa *load balancing* dapat berjalan dengan baik dengan mendistribusikan beban pada *server* yang memiliki waktu respon tercepat.

2.2 *Load Balancing*

Load balancing bisa diartikan sebagai teknik atau proses pendistribusian lalu lintas jaringan dengan menggunakan perangkat-perangkat jaringan. Teknik ini akan mendistribusikan beban *traffic* pada dua atau lebih jalur koneksi secara seimbang. *Load balancing* juga mendistribusikan beban kerja secara merata di dua atau lebih komputer, *link* jaringan, *CPU*, *hard drive* atau sumber daya lainnya, untuk mendapatkan pemanfaatan sumber daya yang optimal (Sirajuddin & Affandi, 2012)



Gambar 2. 1 Sistem *Load Balancing*

Gambar 2.1 adalah gambar layanan *web* yang menggunakan mekanisme *load balancing.* Pada gambar tersebut terdapat 3 *web server* yang akan melayani *request* dari pengguna. *Load balancer* akan mendistribusikan sebuah *traffic* pada *web server* sesuai dengan metode yang digunakan. Dengan demikian metode *load balancing* dapat mencegah terjadinya *overload* pada sebuah *server*, dapat memperkecil waktu respon dan dapat memberikan *throughput* yang maksimal.

2.3 *Software Defined Network*

Softwar Defined Network adalah merupakan sebuah pendekatan baru untuk membangun, mendesain serta me-*manage* jaringan komputer. Konsep dasar *SDN* sendiri adalah melakukan pemisahan antara *control plane* dan *forwarding plane*, dimana *controller* tersebut dapat kita program secara langsung (Kartadie & Utami, 2014). Dengan adanya pemisahan tersebut, maka diharapkan perangkat jaringan dapat di-*manage* melalui *controller* saja. Dalam pengimplementasianya, dibutuhkan sebuah *API* untuk mengoneksikan seluruh perangkat jaringan kedalam sebuah *controller* yang dapat di program sesuai kebutuhan. Dari kebutuhan itulah paradigma *Software Defined Network* muncul, dimana jaringan dapat diatur atau didefinisikan melalui sebuah software.

2.4 *Openflow*

Openflow merupakan protokol yang digunakan oleh *controller* agar dapat melakukan komunikasi dengan infrastruktur jaringan dibawahnya. Openf*low* menjelaskan bagaimana mekanisme komunikasi yang dilakukan antara *controller layer* dan *forwarding layer* (Open Networking Foundation, 2009). Selain itu OpenF*low* juga memberikan akses langsung untuk mengatur dan mengubah *forwarding plane* pada *network device,* baik *physical* maupun *virtual.* Tidak hanya *openflow* satu-satunya protokol yang dapat digunakan untuk mengembangkan *SDN*, masih banyak protokol lain yang dapat digunakan, baik yang *open source* maupun yang berbayar. Ada tiga bagian dalam *swtich* dengan protokol *openflow* dalam SDN, yaitu *Tabel Flow*, *Secure Chanel*, dan *Protokol Openflow.*

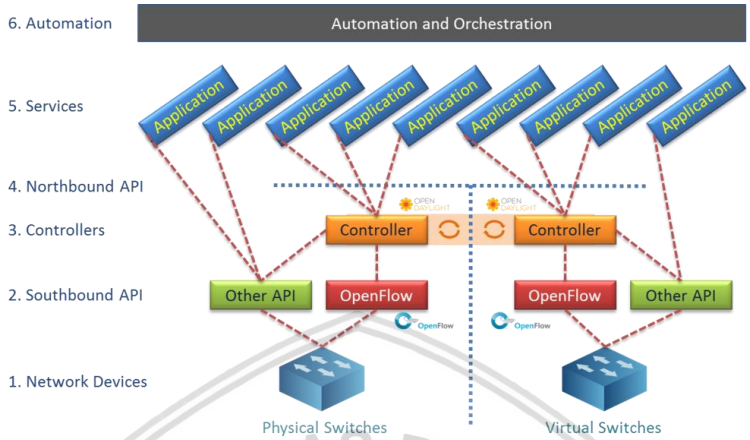
1. *Tabel flow*: Menentukan *action* atau tindakan terhadap suatu paket.
2. *Secure channel*: Interface yang menghubungkan antara *switch* dan *controller* yang menyebabkan terjadinya komunikasi paket dan *flow* antara *controller* dan *switch*.
3. *Protocol openflow*: merupakan suatu protokol yang berfungsi sebagai media komunikasi *switch* dan *controller.*

2.5 *Controller*

Controller merupakan bagian dari *SDN* yang memiliki peran sangat penting. Fungsi dari *controller* adalah mengelola kontrol aliran ke *switch* / *router* (melalui *API*) dan mengatur jalur mana yang akan dipilih. Pengaturan yang ada pada jaringan seperti *routing, load balancing* dan pengaturan lainnya ada pada

controller, sehingga *controller* juga bisa disebut sebagai ”otak” dalam arsitektur jaringan tersebut.

Ada beberapa *controller* yang dapat digunakan pada arsitektur *SDN* yaitu *pyretic*, *POX*, Ryu, dan *Floodlight*.



Gambar 2. 2 *Controller SDN*

Sumber: (SDN Control Plane & SDN Data Plane, 2018)

Gambar 2.2 menjelaskan bahwa *controller* merupakan pusat kendali di dalam arsitektur jaringan *SDN* yang berhubungan dengan *switch* melalui protokol *openflow,* baik dengan *virtual switch* maupun *physical switch*. *Controller* akan mengambil keputusan terhadap paket yang datang dari jaringan. Ada banyak *controller* yang dapat digunakan pada saat ini, baik yang komersil ataupun *open source* yang dapat digunakan pada arsitektur *SDN*. *POX* adalah salah satu *controller* yang ada pada *SDN*. Di dalam *POX* ini terdapat banyak module *library* yang digunakan seperti *forwarding* dan *load balancing*. *POX* memungkinkan *programmer* jaringan dan operator untuk menulis aplikasi jaringan modular dengan singkat dan memberikan abstraksi yang kuat (Lara & Kolasani, 2014).

2.6 Metode *Round Robin*

Metode *round robin* merupakan metode yang akan membagi beban secara merata pada beberapa *server*. Beban akan dibagikan dari *satu server* ke *server* lainnya secara bergiliran dan berurutan. Metode ini tidak memperdulikan kapasitas *server* ataupun beban *request*. Konsep dasar dari metode *round robin* adalah dengan menggunakan *time sharing*, pada intinya metode ini memproses antrian secara bergiliran. Jika terdapat 3 *server* (1,2,3), *request* pertama akan diarahan ke *server* 1, *request* ke dua akan diarahkan ke *server* 2, *request* ke 3 akan diarahkan ke *server* 3 dan *request* ke empat akan diarahkan kembali ke *server* 1, begitu seterusnya tanpa memperdulikan kapasitas pada *server* (Ellrod, Load Balancing – Round Robin, 2010) .

Tabel 2. 1 *PseudoCode* Metode *Round Robin*

Pseudocode Metode *Round Robin*	
1	Begin
2	initialize the lastServer value to 0
3	Determine the totalServer
4	lastServer value is lastServer + 1 mod totalServer
5	Return lastServer
6	End

Tabel 2.1 menjelaskan bahwa proses diawali dengan inisialisai *server* terakhir dengan *index* ke 0. Selanjutnya dilakukan perhitungan total *server*. Kemudian dilakukan perhitungan dengan menambahkan indeks *server* terakhir dengan 1, kemudian dilakukan proses perhitungan modulo dengan jumlah total *server*. Dari hasil perhitungan tersebut merupakan *server* yang akan menerima *request* dari *client*.

2.7 Metode Berbasis *Response Time*

Metode *Berbasis response time* melakukan pembagian beban berdasarkan waktu respon tercepat dari sebuah *server*. Karena waktu respon *server* mencerminkan kemampuan dari *server* tersebut (Zhong & Fang, 2016). Untuk mendapatkan waktu respon dari setiap *server*, *controller* akan mengirimkan *packet_out* ke *switch*, kemudian *switch* akan meneruskan paket tersebut ke *server*. Selanjutnta s*erver* mengirimkan pesan balasan ke alamat sumber yaitu *controller*. Dari pesan balasan tersebut *controller* mendapatkan waktu respon dari setiap *server*.

Tabel 2. 2 *PseudoCode* Metode Berbasis *Response Time*

PseudoCode Metode Berbasis *Response Time*	
1	Begin
2	The controller sends the ARP request packet
3	The controller records the time when the ARP packet is sent
4	The controller receives a packet arp response
5	Controller calculates response time and save to
6	server response’s variable
7	Get minimum response time from server response
8	Save minimum response time to minimum responseTime
9	Return minimum responseTime
10	End

Tabel *pseudocode* 2.2 menjelaskan bahwa proses diawali dengan *contoller* mengirimkan paket *ARP* ke *server* dan mencatat waktu kapan paket tersebut dikirim, selanjutnya *controller* menerima *response* dari *server* dan mencatat waktu kedatangan paket tersebut, dan dilakukan perhitungan nilai *response time* dari *server*. Selanjutnya dilakukan proses *load balancing* dengan memilih nilai *response time* terkecil dari *server*. Jika *controller* tidak mendapatkan informasi berupa waktu respon dari server, maka proses *load balacing* akan diarahkan ke *server* 1 sebagai *default*.

2.8 Metode Berbasis *CPU Usage*

Metode berbasis *CPU* adalah metode yang melakukan proses *load balancing* berdasarkan *CPU resource* yang terkecil (Julianto, 2017). Sebelumnya *server* terlebih dahulu mengirimkan *resource* berupa *CPU usage* ke *controller* secara terus-menerus. Kemudian *controller* melakukan perhitungan dengan membandingan *nilai resource* tersebut. Dari hasil perbandingan tersebut akan dimbil nilai *resource* yang terkecil. Kemudian ketika *client* melakukan *request*, *controller* akan melakukan proses *load balancing* dengan mengarahkan *request* tersebut pada *server* yang memimiliki pemakaian *CPU* terkecil.

Tabel 2. 3 *PseudoCode* Metode *Agent Psutil*

Pseudocode Metode *Agent Psutil*	
1	Begin
2	Get CPU usage information from system using Psutil Module.
3	Store CPU usage information to cpuUsage variable.
4	Connect to Controllerusing UDP socket.
5	While True:
6	Send CPU usage to controller
7	Endwhile
	End

Tabel 2.3 menjelaskan proses pengiriman *CPU usage* ke *controller*. Proses diawali dengan mengambil nilai *CPU usage* dari setiap *server* dengan mengunakan modul *psutil* dan menyimpannya ke sebuah variabel. Kemudian *server* melakukan koneksi dengan *controller* menggunakan protokol *UDP* dan mengirimkan nilai *CPU usage* ke *controler.*

Tabel 2. 4 *PseudoCode* Metode Berbasis *CPU Usage*

PseudoCode Metode Berbasis *CPU Usage*	
1	Begin
2	Resource CPU usage information from web server
3	Store CPU usage information to cpuUsage variable
4	Initilize minimumResource to web server 1
5	If no resource in cpuUsage
6	Return minimumResource
7	Get minimum resource from cpuUsage
8	Set minimium resource to minimumResource
9	Return minimumResource
10	End

Tabel 2.4 menjelaskan tentang proses pemilihan *server* pada metode berbasis *CPU usage* . Proses diawali dengan memeriksa apakah data *CPU usage* tersedia apa tidak. Jika data tidak tersedia, maka *server* 1 akan dipilih sebagai *default.* Jika data tersedia, maka nilai *CPU usage* tersebut akan disimpan di sebuah variabel untuk dilakukan proses perbandingan. Nilai *CPU usage* yang terkecil akan diambil dan ketika ada *request* dari *client* akan diarahkan pada *server* yang memiliki penggunaan *CPU* terkecil.

2.9 *Web Server*

Web server merupakan perangkat lunak dalam *server* yang memberikan layanan berupa permintaan (*request*) kepada pengguna *web* dengan mengunakan protokol *HTTP* atau *HTTPS.* Kemudian *web server* akan mengirimkan kembali *(respon)* hasil dari permintaan tersebut kedalam bentuk halaman web. Penggunaan paling umum dari sebuah *web server* adalah untuk menempatkan sebuah situs *web*, namun pada kenyataannya, kegunaan dapat diperluas sebagai tempat penyimpanan data ataupun untuk menjalankan sejumlah aplikasi bisnis (Kadir, 2003)

Berdasarkan penjelasan yang sudah dipaparkan dapat disimpulkan fungsi dari *web server* adalah untuk mentransfer ataupun untuk memindahkan data yang diminta oleh *user* dengan menggunakan protokol tertentu. *Web server* akan memberikan data yang diminta oleh *user* dengan memberikan respon berupa tampilan halaman *web* yang akan ditampilkan pada *user*.

2.10 *Psutil*

Psutil merupakan modul *library python* yang berfungsi mendapatkan sebuah informasi pada semua proses yang berjalan dari suatu perangkat komputer seperti *CPU, memory, hardisk* dan sebagainya (Psutil Documentation, 2018). *Library* ini dibangun dengan basis bahasa pemrograman *python* dan dapat

digunakan pada berbagai *platform* seperti *Linux, Windows, FreeBSD, OSX* dan lain-lain. Pada penelitian ini *psutil* digunakan untuk mengambil informasi *resource CPU* dari perangkat komputer.

2.11 *Httperf*

HTTP merupakan protokol yang digunakan untuk komunikasi antara *client* dan *server* pada sistem *web*. Untuk mengetahui kinerja dari *Web Server* dibutuhkan alat *(tool)* yang dapat menghasilkan (*generates*) beban kerja *HTTP* pada *server* (Mosberger, 1998). Oleh karena itu, kita dapat mengatasi permasalahan tersebut dengan menggunakan *Httperf*. *Httperf* dapat melakukan *generate* beban kerja dengan protokol *HTTP* pada alamat *host*, *port* maupun jumlah koneksi tertentu.

```
Total: connections 27000 requests 26701 replies 26701 test-duration 179.996 s

Connection rate: 150.0 conn/s (6.7 ms/conn, <=47 concurrent connections)
Connection time [ms]: min 1.1 avg 5.0 max 315.0 median 2.5 stddev 13.0
Connection time [ms]: connect 0.3

Request rate: 148.3 req/s (6.7 ms/req)
Request size [B]: 72.0

Reply rate [replies/s]: min 139.8 avg 148.3 max 150.3 stddev 2.7 (36 samples)
Reply time [ms]: response 4.6 transfer 0.0
Reply size [B]: header 222.0 content 1024.0 footer 0.0 (total 1246.0)
Reply status: 1xx=0 2xx=26701 3xx=0 4xx=0 5xx=0

CPU time [s]: user 55.31 system 124.41 (user 30.7% system 69.1% total 99.8%)
Net I/O: 190.9 KB/s (1.6*10^6 bps)

Errors: total 299 client-timo 299 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

Gambar 2. 3 Contoh *Output* dari *Httperf*

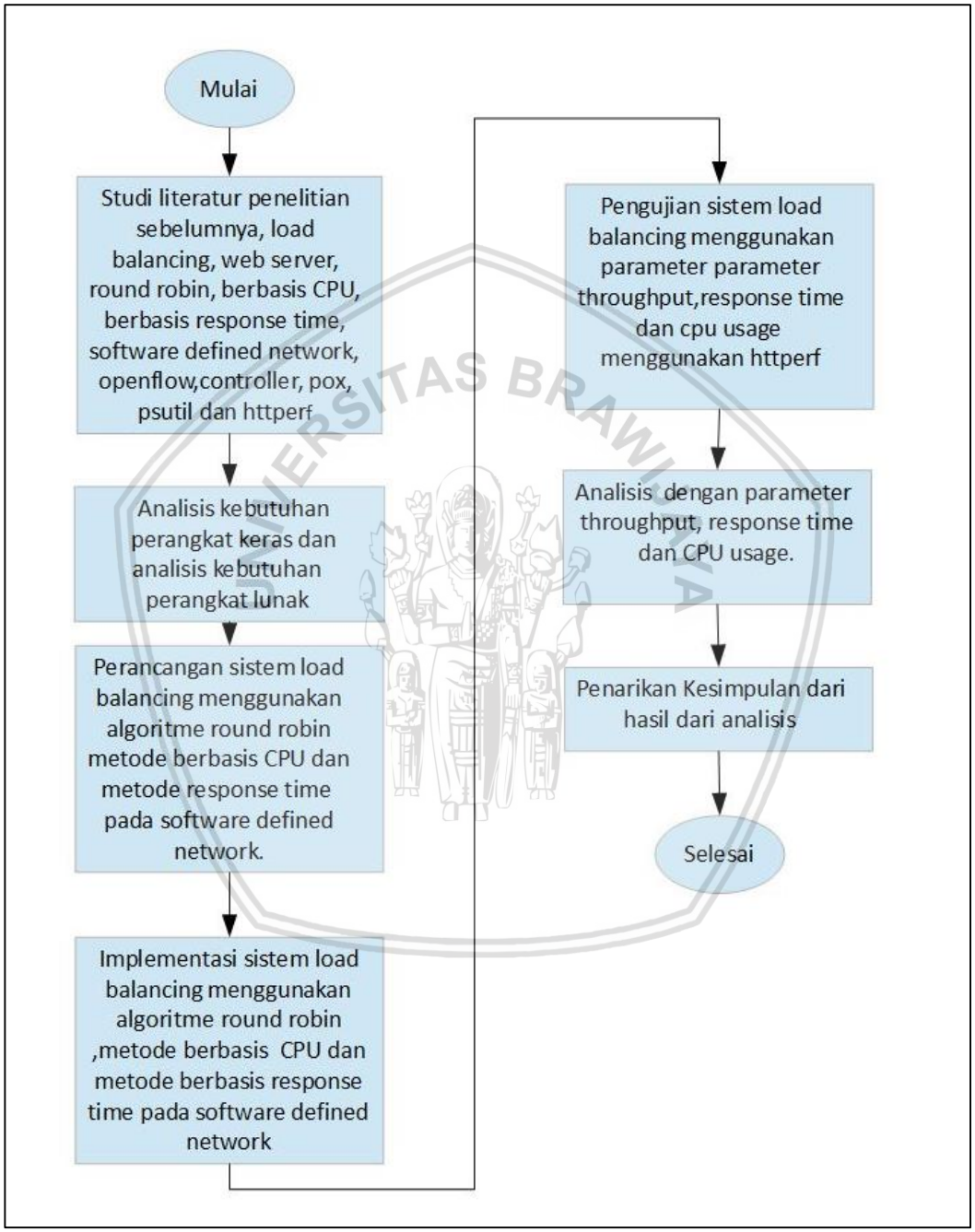
Gambar 2.3 merupakan contoh penguujian menggunakan *Httperf* untuk menguji kinerja *load balancing*, yang dapat dijelaskan sebagai berikut :

- *CPU Time* merupakan pengujian dengan melihat tingkat penggunaan *CPU* dari sisi *user*.
- *Net I/O* merupakan pengujian dengan melihat parameter *throughput* yaitu berapa banyak data yang dapat mengalir dari client ke server dalam satuan KB/s.
- *Reply Time* merupakan pengujian dengan melihat informasi tentang berapa lama waktu untuk *server* untuk merespon dan berapa lama waktu untuk menerima balasan.
- *Total Error* merupakan pengujian dengan melihat ada berapa banyak jumlah paket yang hilang atau *error*.

Pada penelitian ini, *httperf* akan mengambil nilai parameter *throughput* dan *response time* yang kemudian akan dilakukan analisis guna mendapatkan kesimpulan dari kinerja *load balancing* pada *web server*.

BAB 3 METODOLOGI

Pada bagian ini akan dijelaskan tahapan-tahapan yang digunakan dalam proses penelitian dari awal hingga akhir. Berikut merupakan tahapan-tahapan metodologi untuk penelitian ini.



Gambar 3. 1 Diagram Alir Metodologi

Gambar 3.1 menjelaskan metodologi penelitian yang akan dilaksanakan oleh peneliti. Langkah pertama adalah mencari dasar teori baik melalui jurnal, buku, maupun artikel yang berkaitan dengan penelitian. Kemudian dilanjutkan

dengan menganalisis kebutuhan, seperti kebutuhanapa saja yang diperlukan dalam membangun sistem *load balancing.* Kebutuhan yang dimaksud seperti kebutuhan perangkat keras dan kebutuhan perangkat lunak. Selanjutnya melakukan perancangan sebagai dasar dalam pengimplementasian sistem tersebut. Setelah itu, dilakukan pengujian dan analisis hasil untuk dilakukan penarikan kesimpulan terhadap penelitian.

3.1 Studi Literatur

Studi literatur bertujuan sebagai dasar-dasar teori yang digunakan untuk menunjang dalam penelitian ini. Adapun literatur yang digunakan dalam penelitian ini yaitu jurnal, artikel, buku, e-book, dan penelitian sebelumnya yang memiliki keterkaitan dengan penelitian ini. Adapun studi literatur dalam penelitian ini meliputi:

1. *Software defined network (SDN)* merupakan infrastrutur yang memisahkan antara *control plane* dan *data plane. Software defined network* menggunakan protokol *openflow* untuk membuat keduannya bisa berkomunikasi meskipun dalam kondisi terpisah.
2. *Controller* merupakan sebuah "otak" dalam *SDN* dalam artian sebagai pusat kendali dalam menjalankan proses-proses *logic* pada *Software defined network.*
3. *Openflow* merupakan salah satu komponen untuk *SDN. Open flow* digunakan untuk memisahkan *control plane* dan *data plane* dan membuat keduanya dapat berkomunikasi.
4. *Load balancing* merupakan metode pendistribusian beban pada dua atau lebih jalur koneksi agar proses pendistribusian beban dapat lebih optimal dan stabil.
5. Metode *round robin* merupakan pemilihan pada *server* yang bekerja dengan cara membagi beban secara bergiliran dan berurutan dari satu *server* ke *server* lainnya.
6. Metode berbasis *CPU usage* merupakan metode pembagian beban pada *server* dengan menggunakan parameter *usage CPU*. Beban akan didistribusikan pada *server* yang memiliki penggunan *CPU* terkecil.
7. Metode berbasis *response time* merupakan metode pemilihan *server* berdasarkan waktu respon tercepat dari sebuah *server*. Beban akan di distribusikan pada *server* yang memiliki *response time* terkecil.
8. *Web server* merupakan sekumpulan halaman yang memberikan data ataupun transfer data informasi terhadap permintaan yang dilakukan oleh *user*.

3.2 Analisis Kebutuhan

Analisis Kebutuhan menjelaskan kebutuhan apa saja yang diperlukan dalam pengerjaan penelitian ini. Analisis kebutuhan diperlukan agar penelitian ini dapat berjalan dan mendapatkan hasil yang diinginkan. Berikut merupakan kebutuhan yang diperlukan dalam penelitian ini:

3.2.1 Kebutuhan Perangkat Keras

Pada penelitian ini membutuhkan perangkat keras dalam implementasinya. Perangkat keras yang digunakan dalam penelitian ini adalah :

1. Pada penelitian ini membutuhkan 4 buah komputer. Adapun spesifikasi komputer yang digunakan pada penelitian ini adalah sebagai berikut:

- Komputer 1 sebagai *Controller*:
  a. Lenovo 3000 H Series
  b. Intel Dual Core E2220 / 2.4 GHz
  c. 4GB RAM
  d. 320 GB SATA 7200 RPM
- Komputer II sebagai *server* (virtual *server*)
  a. Intel(R) Celeron(R) CPU 1007U @1.50GHz (2 CPUs)
  b. RAM 4 GB
  c. Hardisk kapasitas 500 GB
  d. Monitor 14 inch
- Komputer III sebagai switch
  a. Lenovo 3000 H Series
  b. Intel Dual Core E2220 / 2.4 GHz
  c. 4GB RAM
  d. 320 GB SATA 7200 RPM
- Komputer IV sebagai client
  a. Intel Coleron N4280
  b. RAM 4 GB
  c. Hardisk 500 GB
  d. Monitor 14 inch

2. Kabel UTP

Kabel UTP digunakan sebagai media transmisi. Kabel UTP digunakan karena penggunaannya yang fleksibel dan tidak memerlukan biaya mahal.

3. Kabel *UTP*

Kabel *UTP* digunakan sebagai media transmisi. Kabel UTP digunakan karena penggunaannya yang fleksibel dan tidak memerlukan biaya mahal.

3.2.2 Kebutuhan Perangkat Lunak

Perangkat lunak mempunyai peran sebagai media interaksi yang menghubungkan perangkat keras dengan pengguna komputer. Perangkat lunak juga berfungsi sebagai pemroses data atau perintah untuk menjalankan proses tertentu. Adapun perangkat lunak yang digunakan dalam penelitian ini adalah:

a. Sistem Operasi *Linux Ubuntu LTS*

Linux Ubuntu LTS dipilih karena memiliki utilitas yang memadai dalam melakukan penelitian ini. *Linux Ubuntu LTS* berifat *user friendly* dan didistribusikan sebagai perangkat lunak bebas.

b. Open vSwitch

Open vSwitch adalah sebuah aplikasi yang digunakan untuk membentuk sebuah *virtual switch* pada sistem operasi *linux*. *Open vSwitch* dapat bekerja pada *router* yang menggunakan sistem operasi *linux* (*kernel linux*) seperti *OpenWrt*. *Open vSwitch* juga dapat diimplementasikan pada komputer *desktop* pada umumnya. Jenis *Open vSwitch* yang digunakan pada penelitian ini yaitu *Open vSwitch 2.2.5*.

c. Python

Bahasa pemrograman yang digunakan dalam penelitian ini adalah bahasa pemrograman *python.* Bahasa pemrograman *python* digunakan karena bahasa pemrograman *python* didukung oleh *controller POX.* Versi *python* yang digunakan adalah *python 2.7*. Bahasa *python* merupakan bahasa yang interpretative dan multi guna yang diklaim sebagai bahasa pemrograman yang handal.

d. Virtual Box

Virtual box merupakan perangkat lunak virtualisasi yang dapat digunakan untuk membuat sistem operasi tambahan pada perangkat yang sama. *Virtual box* digunakan untuk membuat *server* vitual dikarenakan adanya keterbatasan *port* yang ada pada *switch* yang tidak memungkinkan untuk penambahan perangkat keras .

e. JetBrain PyCharm

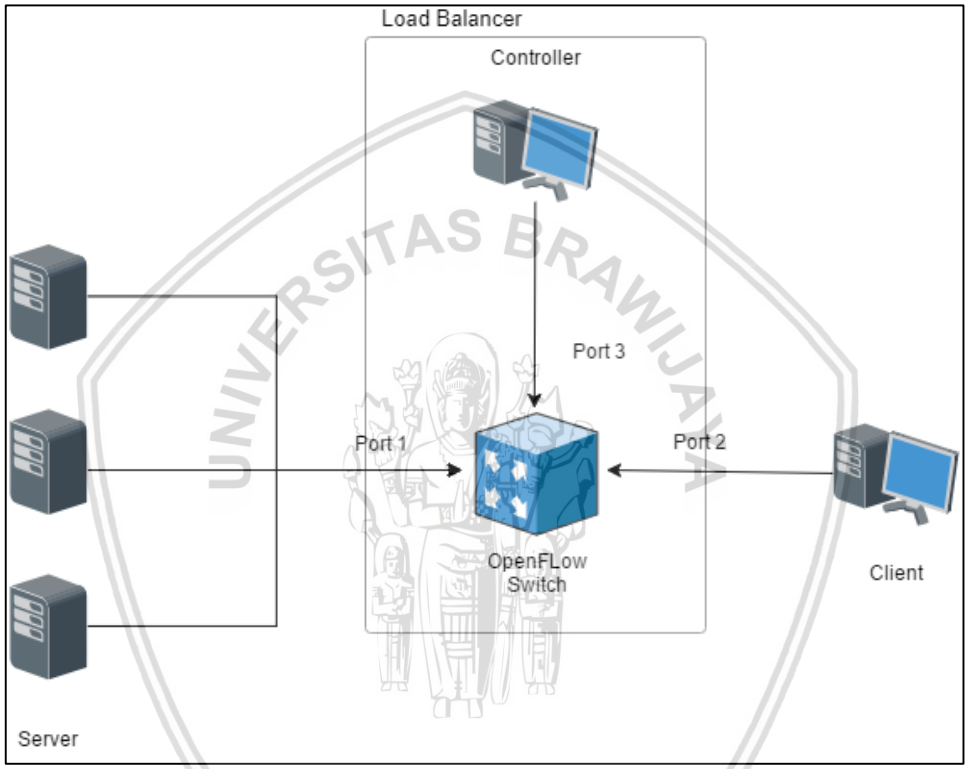
JetBrain PyCharm merupakan text editor yang digunakan untuk melakukan perubahan atau editing pada sebuah kode program. Penggunaan text editor dikarenakan agar dalam proses pengeditan menjadi lebih mudah.

f. Google Chrome

Browser digunakan pada saat melakukan installasi *switch* dan melakukan pengaturan *webserver*. *Browser* juga bisa sebagai indkator bahwa *server* telah hidup dan mampu melayani *request* .

3.3 Perancangan Sistem

Pada tahap ini merupakan tahapan untuk membangun sistem setelah dilakukan studi literatur dan analisis kebutuhan sistem. Perancangan topologi sistem dapat dilihat pada gambar 3.2 yang dijelaskan sebagai berikut.



Gambar 3. 2 Diagram Perancangan Topologi Sistem

Gambar 3.2 merupakan gambaran sebuah sistem dimana di dalam sistem tersebut mengimplementasikan aritektur *SDN* dengan menggunakan tiga buah *server* yang telah terkonfigurasi, satu buah *SDN controller* dan satu buah *SDN switch* dan satu buah *client*.

1. *SDN Switch*

SDN switch yang digunakan adalah *Open vswitch* yang diimplemetasikan pada sebuah komputer menggunakan operasi *Linux Ubuntu Server* 14.04 *LTS*. Selanjutnya *switch* tersebut akan dikonfigurasi untuk meneruskan paket dengan menggunakan tiga *port* yang berbeda. *Port* pertama dengan *interface* eth0 terhubung dengan *server*, *port* kedua dengan *interface* eth1 terhubung dengan *client*, dan ketiga *port* dengan *interface* eth2 terhubung dengan *controller.*

2. *SDN Controller*

Dalam pengimplementasian *SDN controller,* sistem operasi yang digunakan adalah *linux ubuntu Desktop 14.04 LTS.* Pada peneltian ini sistem jaringan yang digunakan adalah *POX-carp* yang dapat diunduh dari *github* resmi *POX*.

3. *Server*

Penelitian ini mengimplementasikan tiga buah *server* virtual yang akan digunakan untuk melayani *request* dari *client*, dimana pada setiap *server* akan berjalan pada sistem operasi *linux Ubuntu dekstop 14.04 LTS* . pada setiap *server* juga akan di *install* sebuah modul tambahan yaitu *psutil* yang digunakan untuk mengambil informasi sumber daya dari perangkat komputer.

4. *Client*

Komputer client akan berjalan pada operasi sistem ubuntu *Server 14.04 LTS.* Selanjutnya client akan di install *httperf* yang digunakan untuk melakukan *request* ke *server.*

3.3.1 Konfigurasi Perangakat

Pada tahap ini menjelaskan proses konfigurasi yang harus dilakukan di setiap perangkat yaitu *server*, *SDN switch*, *SDN controller*, dan *client. S*etiap perangkat yang sudah terkonfigurasi akan saling dihubungkan dengan kabel *UTP* (*Unshielded Twisted Pair*) agar perangkat dapat saling berkomunikasi.

3.3.1.1 Konfigurasi *Server*

Server yang digunakan memiliki spsifikasi *server* sebagai berikut :

Tabel 3. 1 Spesifikasi *Server*

Ciri Beda	**Server 1**	**Server 2**	**Server 3**
Prosesor	Processor 4 core	Processor 2 core	Processor 1 core
Sistem Operasi	Ubuntu Server 14.04 LTS	Ubuntu Server 14.04 LTS	Ubuntu Server 14.04 LTS
Memory	RAM 1 GB	RAM 1 GB	RAM 1 GB
IP Adrdess	192.168.1.11	192.168.1.12	192.168.1.13

Server yang digunakan adalah 3 buah *server* virtual untuk melayani *request* dari *client*. Setiap *server* akan dikonfigurasi memiliki jumlah *core* yang berbeda-beda dengan tujuan untuk mengetaui apakah penerapan metode berjalan dengan baik atau tidak. Semua *server* berjalan pada sistem operasi ubuntu server *14.04 LTS.* Pengalamtan IP yang digunakan yaitu jenis *IP address* kelas C. karena pada penelitian ini hanya menggunakan 4 *host*. Karena jenis *IP address* kelas C digunakan untuk jaringan dengan jumlah *host* yang kecil, maksimal 254 host.

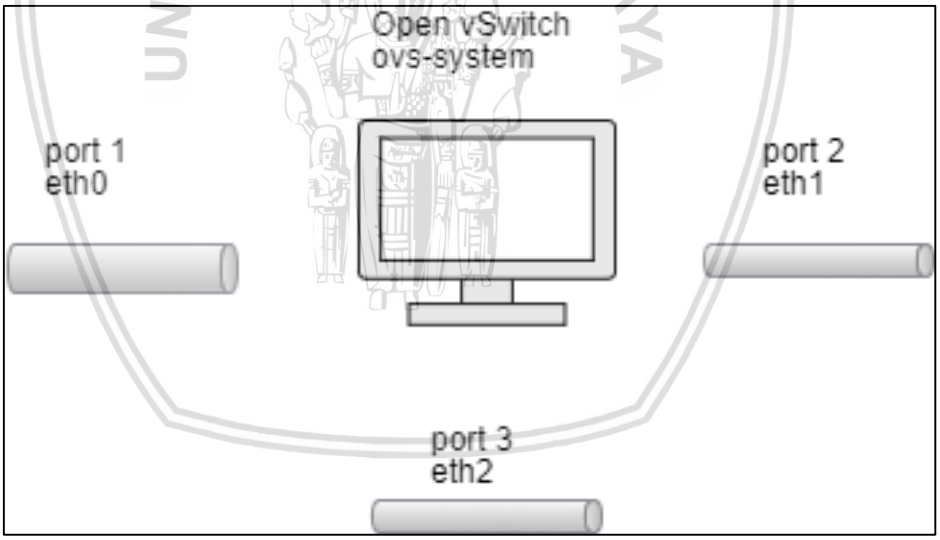
3.3.1.2 Konfigurasi *SDN Switch*

Pada konfigurasi *switch* mengunakan *Open vSwitch 2.2.5* yang di *install* pada sebuah komputer dengan spesifikasi yang sudah dijelaskan pada subbab kebutuhan perangkat keras sebelumnya. Aplikasi *Open vSwitch* akan dikembangkan pada sebuah komputer yang berjalan pada sistem operasi *ubuntu dekstop 14.04 LTS*. Komputer memiliki tiga buah *network interface* yang nantinya akan digunakan sebagai *port* dari *Open vSwitch*. Tabel berikut menjelaskan pemetaan anatara *port Open vSwitch* dan *network interface.*

Tabel 3. 2 Spesifikasi *Switch*

Interface	Port	Tujuan
eth0	Port 1	Server cluster
eth1	Port 2	Client
eth2	Port 3	Controller

Pada masing-masing *port* akan bertanggung jawab dalam meneruskan paket sesuai dengan aturan yang sudah terkonfigurasi pada *flow-table.* Pada gambar 3.3 di bawah ini menjelaskan topologi dari *Open vSwitch.*



Gambar 3. 3 Topologi *Open vSwitch*

3.3.1.3 Konfigurasi *SDN Controller*

Controller akan dikonfigurasi pada komputer yang berjalan pada *ubuntu dekstop 14.04 LTS*. *IP address controller* akan dikonfigurasi memiliki *IP address* 192.168.1.10. Pada penelitian ini *load balancing* akan dibuat dengan mengunakan *controller POX. Controller* akan bekerja sebagai pusat kendali dalam arsitektur *SDN*. *Controller* akan memandu *switch* dalam meneruskan paket dan pada saat yang sama menjalankan *load balancing* sesuai dengan metode yang digunakan.

Dalam proses menjalankan c*ontroller* dibutuhkan sebuah *IP sevice* yaitu 192.168.1.100 yang diakses oleh *server* dan *client* untuk berkomunikasi. *IP service* digunakan agar *IP address* asli dari *controller* tidak diketahui oleh selain *controller* itu sendiri.

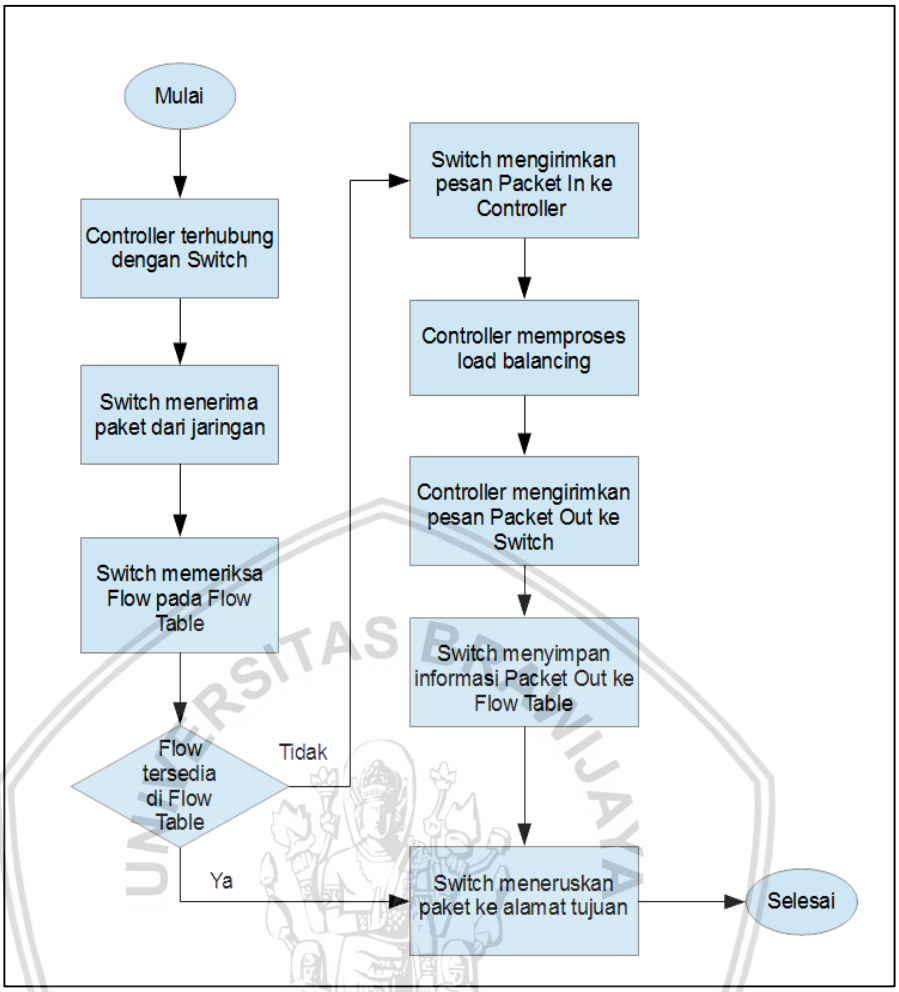
3.3.1.4 Konfigurasi *Client*

Aplikasi yang digunkan oleh *client* untuk melakukan pengujian adalah *httperf*. Selanjutnya *client* akan dikonfigurasi dengan alamat *IP address* yaitu 192.168.1.15. *client* akan melakukan beberapa *request* pada *server* dengan menggunakan *httperf* untuk mengetahui kinerja dari jaringan sesuai dengan parameter yang telah ditentukan.

3.3.2 Perancangan *Load Balancing*

3.3.2.1 Alur Sistem Secara Umum

Alur sistem secara umum menjelaskan proses kerja dari komponen-komponen sistem dari awal sampai akhir dalam melakukan proses *load balancing*. Proses diawali dengan datangnya *traffic* dari jaringan sampai penentuan aksi oleh *controller* terhadap *traffic* tersebut. Untuk lebih jelas dapat dilihat pada gambar 3.4.



Gambar 3. 4 *Flowchart* Alur Sistem Secara Umum

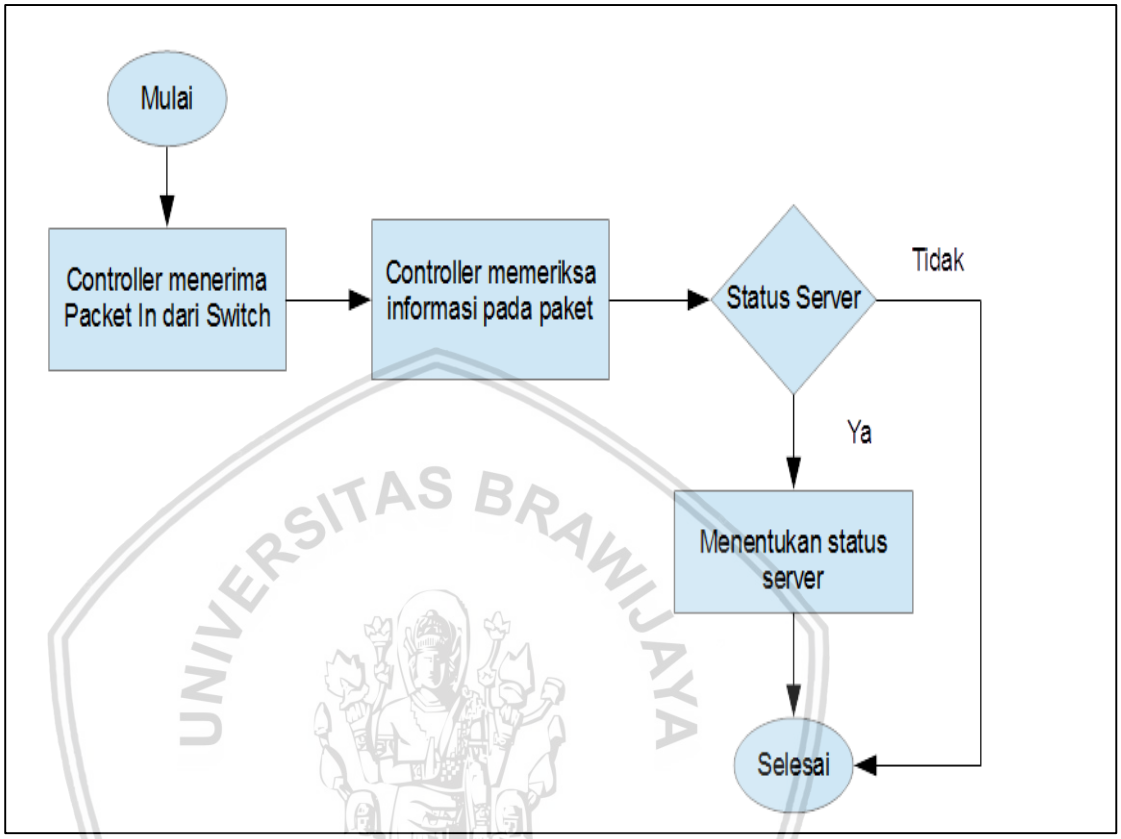
Gambar 3.4 menjelaskan bahwa alur sistem *load balancing* diawali dengan terhubungnya *switch* ke *controller*. Kemudian *switch* akan menerima paket yang masuk ke sistem. Sebelum mengirimkan *packet_In* ke *controller*, terlebih dahulu *switch* memeriksa *entri flow* pada *flow table.* Jika *entry flow* tidak tersedia, maka *Switch* akan mengirimkan pesan *packet_In* ke *controller* dan *controller* akan memproses *load balancing* dengan menentukan alamat tujuan dari paket tersebut berdasarkan *server* dengan jumlah koneksi paling sedikit. Setelah *Switch* menerima pesan packet_Out dari *controller*, *switch* akan menyimpan *entry* ke *flow* table. Sehingga jika terdapat paket dengan informasi yang sama, *switch* tidak mengirimkan pesan *packet_In* ke *controller*. Setelah itu paket akan diteruskan berdasarkan alamat tujuan dari paket tersebut.

3.3.2.2 Alur Sistem *Load Balancing*

Alur sistem *load balancing* ini merupakan mekanisme pembagian beban yang terdapat pada *controller.* Proses pembagian beban dilakukan berdasarkan metode *round robin* , metode berbasis *response time* dan metode berbasis *CPU*

usage. Didalam mekanisme pembagian beban terdapat alur pemeriksaan status *server*, pemeriksaan paket dari *client* dan pemeriksaan paket dari *server*.

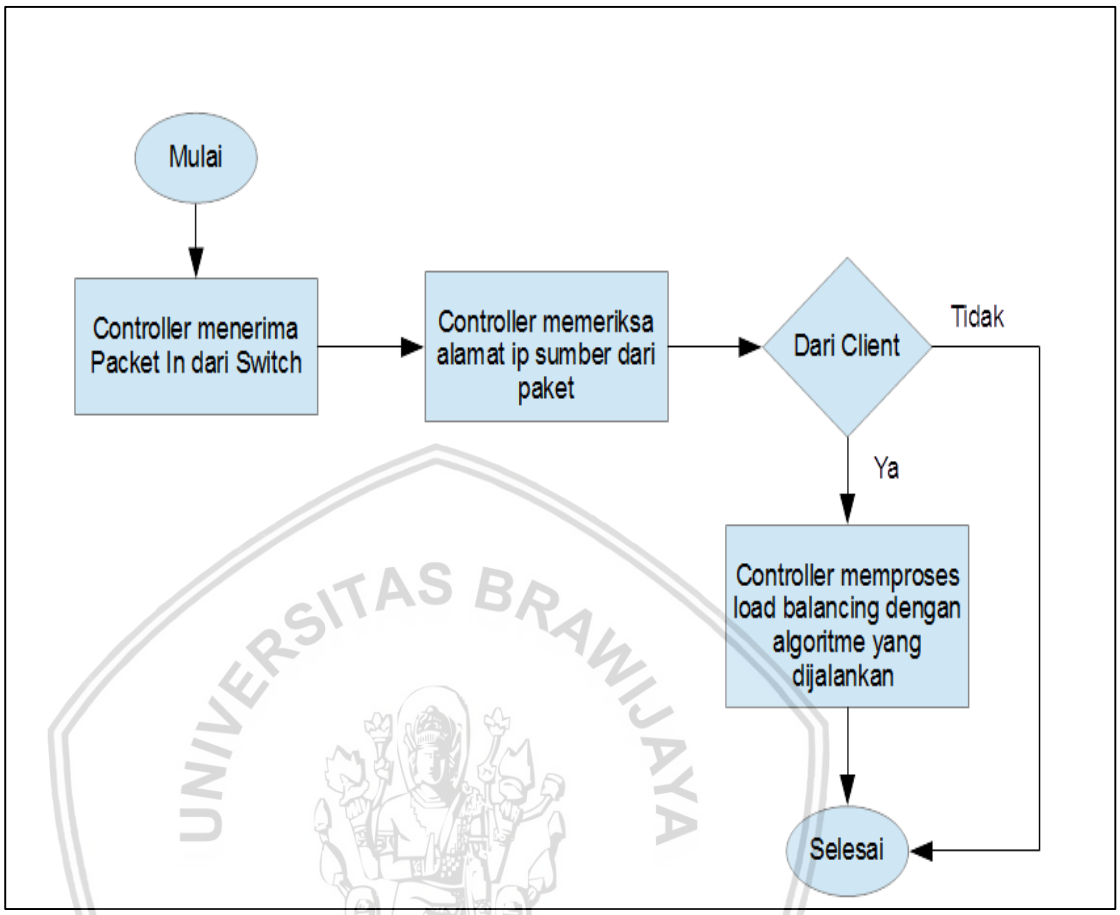
a. Alur Pemeriksaan Status *Server*



Gambar 3. 5 Alur Pemeriksaan Status *Server*

Gambar 3.5 merupakan tahap-tahap pemeriksaan informasi status dari *server* dengan memanfaatkan *packet_In* pada c*ontroller*. Paket ini akan dikirimkan oleh setiap *server* setelah controller mengirim *ARP-REQUEST* dengan memberi balasan berupa *ARP-REPLAY*. Jika *server* yang terdaftar didalam *cluster* mengirimkan paket dengan jenis ini, maka *server* tersebut dianggap sedang beroperasi atau sedang *up.* Namun, jika *server* tidak mengirimkannya *server* akan dianggap dalam kondisi *down*.

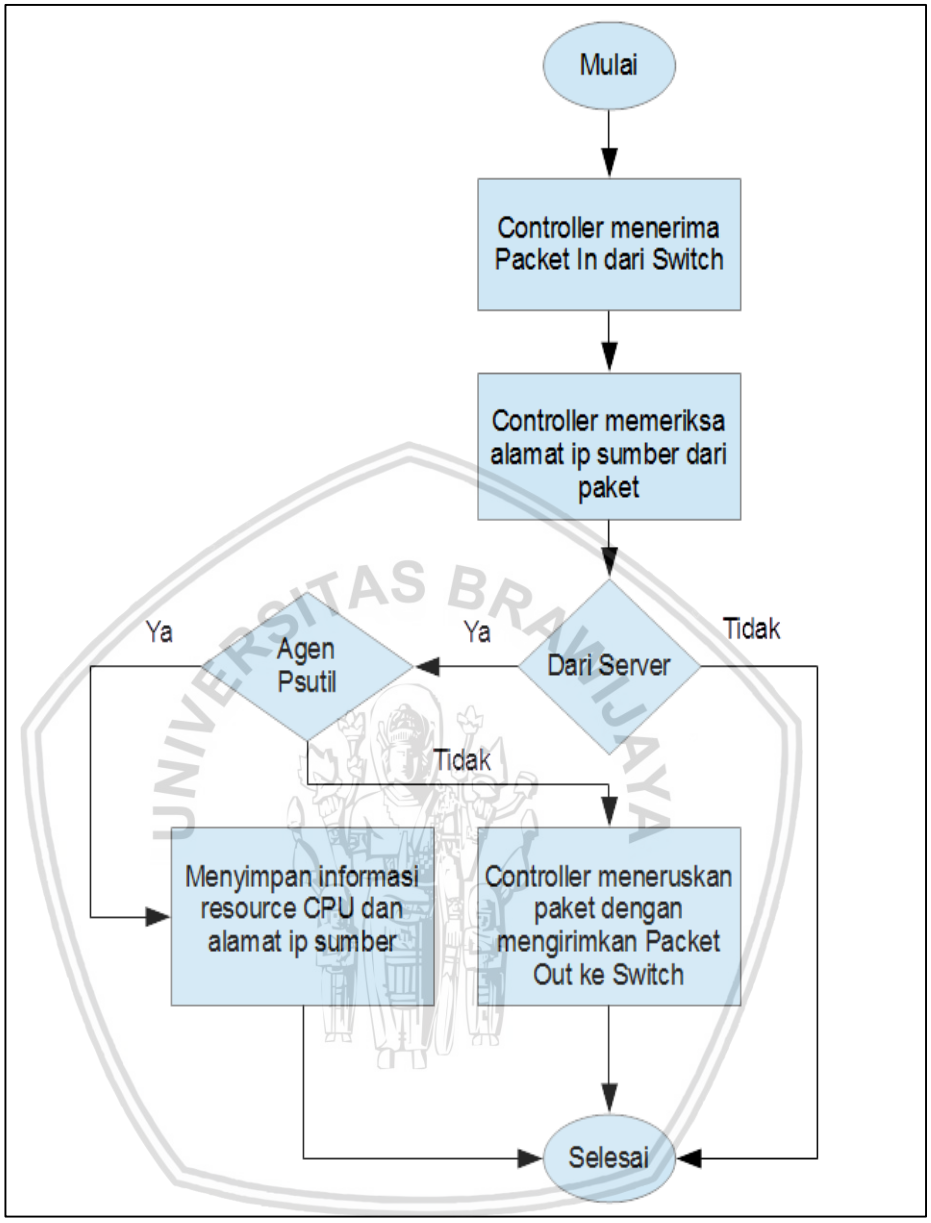
b. Alur Pemeriksaan Paket dari *Client*



Gambar 3. 6 Alur Pemeriksaan Paket dari Client

Gambar 3.6 merupakan proses *load balancing* yang dilakukan jika paket tersebut datang dari c*lient* dengan memeriksa alamat *IP* sumber paket dan alamat *IP* tujuan dari paket tersebut. Selanjutnya *controller* akan melakukan proses *load balancing* sesuai dengan metode yang dijalankan.

c. Alur Pemeriksaan Paket dari *Server*



Gambar 3. 7 Alur Pemeriksaan Paket dari *Server*

Gambar 3.7 menjelaskan alur pemeriksaan paket dari *server.* terdapat beberapa jenis paket yang dikirimkan oleh *server* yaitu paket dengan *IP* tujuan ke c*lient* dan paket dari *Agen Psutils*. Jika paket tersebut merupakan paket dari *Agen Psutils* pada *server* dan dikirim menggunakan protokol *UDP,* maka *controller* akan menyimpan informasi dari *Agen Psutils*. Namun, jika bukan dari *Agen Psutils,* maka *controller* akan meneruskan paket dengan mengirimkan Paket *Out* ke *Switch* untuk diteruskan ke alamat tujuan dari paket.

3.4 Implementasi Sistem

Perancangan sistem dibuat berdasarkan kebutuhan dan perancangan sistem yang telah dibuat sebelumnya. Adapun implementasi sistem sebagai berikut:

- Konfigurasi perangkat
- Implementasi metode *round robin*, metode berbasis *CPU usage* dan metode berbasis *response time*

3.4.1 Konfigurasi Perangkat

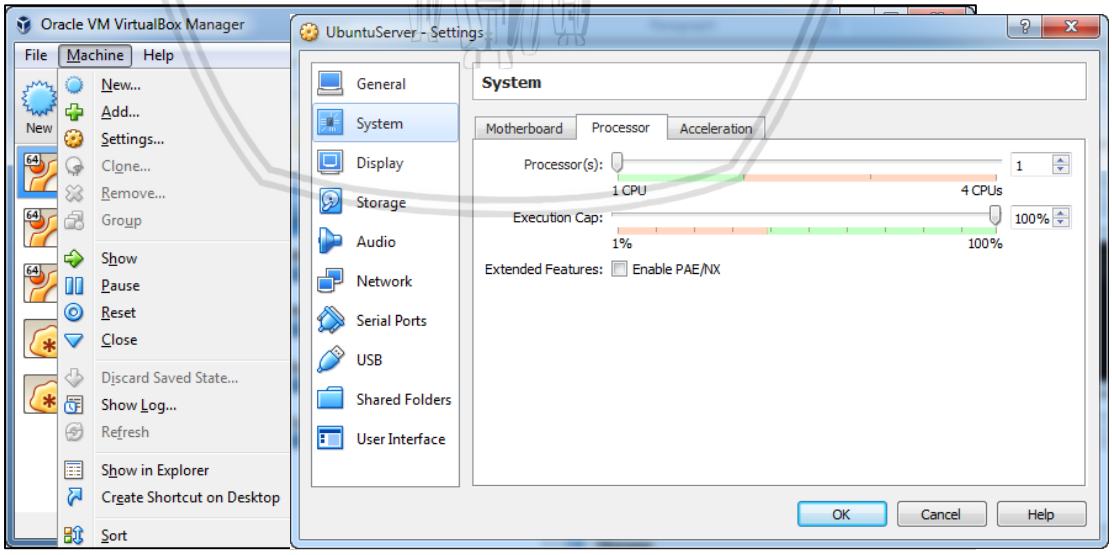
Pada bab sebelumnya sudah dijelaskan konsep perancangan yang akan digunakan sebagai dasar pada proses implementasi. Proses implemestasi akan disesuaikan dengan konsep perancangan yang sudah dijelaskan sebelumnya. Pada tahap ini akan dilakukan konfigurasi pada setiap perangkat.

3.4.1.1 Konfigurasi *Server*

Server yang digunakan dalam penelitian ini adalah tiga *server* virtual menggunakan *software virtual Box* . setelah dilakukan instalasi *server* kemudian dilakukan beberapa konfigurasi seperti menentukan jumlah *core* pada masing-masing *server* dan pengalamatan *IP address* di setiap *server*.

1. Pengaturan *Server*

   Setiap *server* memiliki jumlah *core* yang berbeda. *Server* 1 di *setting* dengan jumlah *core* 4, *server* 2 dengan jumlah *core* 2 dan *server* 3 di *setting* dengan jumlah *core* 1.



Gambar 3. 8 Konfigurasi *Server* Virtual

Gambar 3.8 menjelaskan proses pengaturan *core* pada setiap *server*. Langkah yang harus dilakukan adalah pilih menu *machine* pada tampilan utama, kemudian pilih *setting*, kemudian pilih *system* dan pilih *prosessor*. Selanjutnya *setting* jumlah *core* pada masing-masing *server* sesuai dengan perancangan yang telah dibuat yaitu 4,2,dan 1 *core*.

2. Konfigurasi *IP Address* pada *Server*

Setelah sebelumnya melakukan pengaturan jumlah *core* pada *server*, kemudian dilakukan pengalamatan *IP address* dari masing-masing *server* virtual. Pada sistem operasi ubuntu pengaturan *IP address* berada pada pada */etc/network/interfaces*. Perintah yang digunakan dalam pengaturan *IP address* adalah sebagai berikut :

```
$sudo nano /etc/network/interfaces
```

Kemudian melakukan konfigurasi sebagai berikut.

```
auto eth0
iface eth0 inet static
    address 192.168.1.11
    netmask 255.255.255.0
    gateway 192.168.1.1
```

Untuk mengetahui apakah konfigurasi yang sudah dilakukan telah berjalan dapat dilihat menggunakan perintah sebagai berikut.

```
$ sudo ifconfig
```

Perintah tersebut akan menampilkan *interface* dari *server*.

```
abdi@abdilaxer:~$ ifconfig
enp3s0f2  Link encap:Ethernet  HWaddr bc:ee:7b:bc:76:8f
          inet addr:192.168.1.11  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::b77d:924a:1c40:69fd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:138 errors:0 dropped:0 overruns:0 frame:0
          TX packets:43 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14359 (14.3 KB)  TX bytes:5239 (5.2 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:231 errors:0 dropped:0 overruns:0 frame:0
          TX packets:231 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:16805 (16.8 KB)  TX bytes:16805 (16.8 KB)
```

Gambar 3. 9 *Interface Server*

Proses yang sudah dijelaskan tersebut kemudian akan dilakukan pada masing-masing *server* dengan *IP address* yang berbeda.

- *Server* 1 dengan alamat IP 192.168.1.11
- *Server* 2 dengan alamat IP 192.168.1.12

- *server* 3 dengan alamat IP 192.168.1.13

3.4.1.2 Konfigurasi *Client*

Pada *client* menggunakan aplikasi *httperf* untuk melakukan pengujian. *Client* akan mengirimkan beberapa *request* kepada *server* dengan menggguakan aplikasi *httperf*. Perintah yang digunakan untuk menginstal *httperf* adalah sebagai berikut.

```
$ sudo apt-get install httperf
```

Setelah melakukan proses instalasi dilakukan pengaturan alamat *IP address* pada *client* agar *client* dapat terhubung dengan *Open vSwitch*. Perintah yang digunakan sebagai berikut.

```
$ sudo nano /etc/network/interfaces
```

Kemudian melakukan konfigurasi sebagai berikut.

```
auto eth0
iface eth0 inet static
    address 192.168.1.5
    netmask 255.255.255.0
    gateway 192.168.1.1
```

Setelah konfigurasi dilakukan, *client* dapat melakukan *request* dengan menggunakan *httperf* dengan perintah sebagai berikut.

```
$ httperf –server [ip service] --port 80 --num-conns a --rate b
```

Dengan perintah tersebut *client* dapat melakukan *request* dengan memberikan banyak koneksi dan *rate* sesuai dengan kebutuhan dalam pengujian.

3.4.1.3 Konfigurasi *SDN Switch*

Pada penelitian ini *switch* yang digunakan adalah *Open vSwitch* yang di *install* pada sebuah komputer. Untuk menginstal *Open vSwitch* perintah yang digunakan adalah sebagai berikut.

```
$ sudo apt-get install openvswitch
```

Setelah *Open vSwitch* terinstall, maka penambahan *bridge* br0 dilakuakan. *Bridge* tersebut bekerja sebagai *virtual switch* yang meneruskan paket sesuai dengan *flow-tabel* yang diberikan. Penambahan *bridge* dapat dilakukan dengan perintah sebagai berikut.

```
$ sudo ovs-vsctl add-br br0
```

Setelah penambahan *bridge* selesai dilakukan, selanjutnya dapat melakukan konfigurasi *Openflow* yang digunakan dan juga dapat menambahkan *controller* pada switch. *Controller* dapat ditambahkan dengan memberikan alamat ip dari komputer yang diatur sebagai *controller* serta protokol yang digunakan. Perintah yang digunakan tersebut adalah sebagai berikut.

```
$ sudo ovs-vsctl set bridge br0 protocols=OpenFlow13
$ sudo ovs-vsctl set-server br0 tcp:10.0.0.5:6633
```

Setelah proses konfigurasi *bridge* selesai selanjutnya dapat melakukan penambahan *port* yang akan digunakan oleh *switch* untuk meneruskan paket. *Port* yang dibuat tersebut harus memilki *interface*, agar paket dapat diteruskan *host* menuju *Open vSwitch* ataupun sebaliknya. Perintah yang dapat dilakukan adalah sebagai berikut.

```
auto eth0
iface eth0 net static
auto eth1
iface eth1 inet static
auto eth2
iface eth2 inet static
auto br0
iface br0
    address 192.168.1.20
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
```

Setelah penambahan *port* selesai dilakukan, selanjutnya *port* tersebut disambungkan ke *bridge* yang telah ditambahkan sebelumnya. *Bridge* tersebut memiliki ip 192.168.1.20. perintah yang dapat dilakukan adalah sebagai berikut.

```
$ sudo su
$ ipvlink set eth0 up
$ ip lin kset eth1 up
$ ip link set eth2 up
$ ovs-vsctl add-port br0 eth0
$ ovs-vsctl add-port br0 eth1
$ ovs-vsctl add-port br0 eth2
```

Selanjutnya untuk melihat *port* tersebut dapat dilakukan perintah sebagai berikut.

```
$ sudo ovs-vsctl show
```

Perintah tersebut akan menampilkan konfigurasi sebagai berikut.

```
abdi@abdilaxer:~$ sudo ovs-vsctl show
f3d3a02b-386e-423d-bb93-a13251cec369
    Bridge "br0"
        Controller "tcp:192.168.1.10"
        Port "eth1"
            Interface "eth1"
        Port "br0"
            Interface "br0"
                type: internal
        Port "eth0"
            Interface "eth0"
        Port "eth2"
            Interface "eth2"
    ovs_version: "2.5.2"
```

Gambar 3. 10 Konfigurasi *Open vSwitch*

Gambar 3.10 menjelaskan *port* yang sudah dibuat sudah terikat dengan sebuah *interface* dan sudah terdapat sebuah *controller* yang sudah di *setting* sebelumnya.

3.4.1.4 Konfigurasi *SDN Controller*

Setelah melakakukan konfigurasi pada *Open vSwitch* dan berjalan lancar, selanjutnya dapat melakukan konfigurasi pada *controller* dengan melakukan instalasi modul *pox* pada perangkat yang digunakan sebagai *controller*. Perintah yang digunakan adalah sebagai berikut.

```
$ git clone http://github.com/noxrepo/pox
```

Kemudian dilakukan konfigurasi sebagai berikut.

```
auto eth0
iface eth0 inet static
    address 192.168.1.10
    netmask 255.255.255.0
    gateway 192.168.1.1
```

Setelah melakukan konfigurasi terebut dan berjalan lancar, *controller* dapat dijalankan dengan menggunakan metode yang dipakai dalam penelitian ini.

3.4.2 Implementasi Metode *Load Balancing*

Metode yang digunakan dalam penelitian ini yaitu metode *round robin,* metode berbasis *CPU usage* dan metode berbasis *response time.* Metode-metode tersebut akan dijalankan pada *controller* dan ketika *client* melakukan *request* akan di *direct* ke *server* sesuai dengan alur masing-masing metode tersebut. Masing-masing memiliki proses *load balancing* yang berbeda-beda. Pada metode *round robin* proses *load balancing* akan diarahkan ke *server* secara bergantian, sedangkan metode berbasis *CPU usage* proses *load balancing* akan diarahkan ke *server* yang memiliki penggunaan CPU terkecil, dan pada metode berbasis *response time* proses *load balancing* akan diarahkan pada *server* yang memiliki respon tercepat.

3.4.2.1 Pemeriksaan Status dari *Server*

Pengiriman paket *ARP* dilakukan guna mengetahui kondisi server dalam keadaan *up* atau *down.* Metode *round robin*, metode berbasis *CPU usage, dan* metode berbasis *response time* memiliki proses yang sama dalam pengiriman paket *ARP.* Ketika *controller* sudah dalam kondisi hidup, *controller* akan malakukan pengiriman paket ke *server.* Kemudian setelah ada balasan dalam bentuk *ARP-REPLAY* dan diketahui bahwa balasan tersebut dari *server,* maka dapat dinyatan bahwa *sever* dalam kondisi *up* dan akan memberi *statement "up"* di *controller.* Berikut *source code* untuk pengecekan paket *ARP*.

Tabel 3. 3 *Source Code* pengiriman *ARP-REQUEST*

Source Code pengiriman *ARP-REQUEST*

```
1  def _do_probe (self):
2      self._do_expire()
3
4      server = self.servers.pop(0)
5      self.servers.append(server)
6
7      r = arp()
8      r.hwtype = r.HW_TYPE_ETHERNET
9      r.prototype = r.PROTO_TYPE_IP
10     r.opcode = r.REQUEST
11     r.hwdst = ETHER_BROADCAST
12     r.protodst = server
13     r.hwsrc = self.mac
14     r.protosrc = self.service_ip
15     e = ethernet(type=ethernet.ARP_TYPE, src=self.mac,
16                  dst=ETHER_BROADCAST)
17     e.set_payload(r)
18     #self.log.debug("ARPing for %s", server)
19     msg = of.ofp_packet_out()
20     msg.data = e.pack()
21     msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
22
23     msg.in_port = of.OFPP_NONE
24     self.con.send(msg)
25     self.arp_timeout tstanding_probes[server] = time.time() +
       self.arp_timeout
```

Tabel 3.3 merupakan kode pengiriman *ARP-REQUEST* yang dapat dijelaskan sebagai berikut :

- Baris 1 merupakan nama *method* untuk mengirim *ARP-REQUEST.*
- Baris 2 merupakan proses pemanggilan fungsi untuk melakukan pengecekan apakah server *time out* atau tidak.
- Baris 4 sampai 5 merupakan proses mengambil *IP server* untuk dilakukan proses pengiriman *ARP-REQUEST.* Pengiriman dilakukan kepada *server* secara bergantian.
- Baris 7 sampai 23 merupakan konstruksi pengiriman *ARP,* antara lain : Jenis paket yang dikirim *ARP-REQUEST,* jenis pengiriman *broadcast,* tujuan ke *server*, *mac address, IP server,* jenis paket *out,* dsb.

- Baris 24 sampai 25 merupakan penentuan waktu *expire* degan perhitungan waktu sekarang + *ARP time out.*

3.4.2.2 Pengecekan Status *Server*

Pengecekan status *server* dilakukan dengan melakukan pengecekan paket terlebih dahulu oleh *controller*. Jika *controller* menerima paket berupa *ARP-REPLAY, controller* akan memproses paket tersebut dan menyatakan bahwa *server* dalam keadaan *up.* Batas maksimal *server* memberi balasan berupa *ARP-REPLAY* adalah 3 detik, jika lebih dari itu *server* akan dinyatakan *down*.

Tabel 3. 4 S*ource Code* Pengecekan Status *Server*

Source Code* Pengecekan Status *server	
1	`def _do_expire (self):`
2	
3	`    t = time.time()`
4	
5	`    for ip,expire_at in self.outstanding_probes.items():`
6	`      if t > expire_at:`
7	`        self.outstanding_probes.pop(ip, None)`
8	`        if ip in self.live_servers:`
9	`          self.log.warn("Server %s down", ip)`
10	
11	`if not tcpp:`
12	`    arpp = packet.find('arp')`
13	`    if arpp:`
14	`        if arpp.opcode == arpp.REPLY:`
15	`            if arpp.protosrc in self.outstanding_probes:`
16	`                del self.outstanding_probes[arpp.protosrc]`
17	`                if (self.live_servers.get(arpp.protosrc, (None,`
18	`None))== (arpp.hwsrc, inport)):`
19	`                    pass`
20	`                else:`
21	`                    self.live_servers[arpp.protosrc] = arpp.hwsrc,`
22	`inport`
23	`                    self.log.info("Server %s up", arpp.protosrc)`
24	`        return`
25	`    return drop()`
26	

Tabel 3.4 merupakan kode pengecekan status *server* yang dapat dijelaskan sebagai berikut :

- Baris 1 merupakan *method* yang digunakan untuk melihat apakah *server* dalam keadaan *down.*
- Baris 3 merupakan inisialisasi bahwa t = waktu sekarang.
- Baris 5 sampai 9 merupakan perbandingan yang digunakan untuk menentukan apakah *server* dalam kondisi *down* apa tidak. Dengan melakukan komputasi apakah waktu sekarang lebih besar dari waktu *expire.* Jika iya, *server* akan dinyatakan dalam kondisi *down.*
- Baris 11 sampai 14 merupakan jika paket bukan *TCP* melainkan berupa *ARP-REPLAY* dilakukan *statement* didalamnya.

- Baris 15 sampai 16 merupakan jika *server* memberi balasan, maka *outstanding_probes* akan dihapus.
- Baris 23 merupakan *update* kodisi *server* dan *IP* pada *dictionary live_server* serta menampilkan *server* dalam kondisi *up.*

3.4.2.3 Pengecekan Paket *Request* dari *Client*

Controller akan melakukan pengecekan pada setiap paket yang datang. Pertama kali yang dilakukan *controller* adalah melakukan *parsing header* untuk menentukan jenis paket yang masuk. Jika paket tersebut diketahui dari *client*, maka akan dilakukan prose *load balancing.*

Tabel 3. 5 *Source Code* Switch Menangani Paket

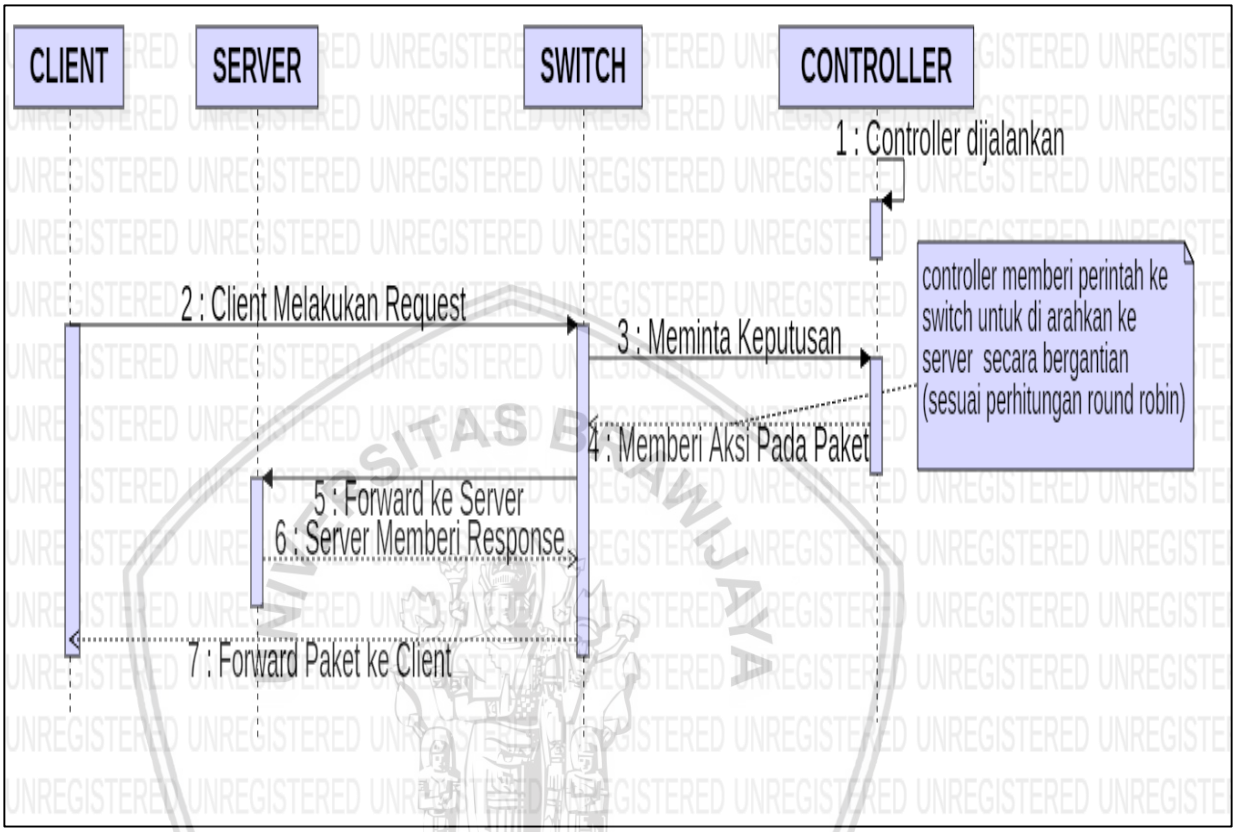
Source Code Switch Menangani Paket	
1	ipp = packet.find('ipv4')
2	
3	elif ipp.dstip == self.service_ip:
4	
5	key = ipp.srcip,ipp.dstip,tcpp.srcport,tcpp.dstport
6	entry = self.memory.get(key)
7	if entry is None or entry.server not in self.live_servers:
8	# Don't know it (hopefully it's new!)
9	if len(self.live_servers) == 0:
10	self.log.warn("No servers!")
11	return drop()
12	
13	#memilih server berdasarkan metode round robin
14	server = self._pick_server(key, inport)
15	# memilih server berdasarkan response time terkecil
16	server = self._pick_server(key, inport)
17	# memilih server berdasarkan metode CPU
18	server = self.resource_algorithm()
19	self.log.debug("Directing to %s", server)
20	
21	entry = MemoryEntry(server, packet, inport)
22	self.memory[entry.key1] = entry
23	self.memory[entry.key2] = entry
24	

Tabel 3.5 merupakan kode switch menangani paket yang dapat dijelaskan sebagai berikut :

- Baris 1 menjelaskan ketika ada paket yang datang dan *IP* adalah *Ipv4* akan dilakukan proses berikutnya.
- Baris 3 sampai 6 menjelaskan jika ada paket datang akan dilakukan pengecekan apakah *IP* tujuan sama dengan *IP* publik , jika sama akan dilakukan *flow* pada *memory.*
- Baris 7 sampai 11 dilakukan pengecekan *flow table,* jika tidak terdapat pada *flow table* maka paket akan di *drop.*
- Baris 13 sampai 19 melakukan proses *load balancing.*
- Baris 21 sampai 23 menyimpan flow didalam *memory.*

3.4.2.4 Metode *Round Robin*

Gambar 3.11 menjelaskan bagaimana proses aliran data dari metode *round robin.*



Gambar 3. 11 *Data Flow* Metode *Round Robin*

Gambar 3.11 merupakan gambar *data flow metode berbasis CPU usage* secara keseluruhan. Proses dibagi menjadi beberapa langkah, yaitu :

1. Menjalankan *Controller*

Pertama yang harus dilakukan adalah menjalankan *controller*. Dalam menjalankan *controller* harus menjalankan *file* yang berisi metode *round robin ,* kemudian mendefinisikan *IP virtual* yang diakses *client* dan *server*. *Controller* juga harus dapat mendeteksi sebuah *server* dalam kondisi hidup atau tidak dengan mengirimkan paket *ARP*. *Controller* akan melakukan pengecekan kondisi *up* atau *down* dengan melakukan perhitungan seperti yang dijelaskan pada tabel 3.3 dan tabel 3.4 yang didapatkan dari pengiriman paket *ARP* ke *server* sampai *server* memberi balasan dalm bentuk *ARP-REPLAY* ke *controller*.

2. *Client* melakukan *request* dengan mengakses *IP service*

Client melakukan *request* menggunakan *httperf* dengan mengakses *IP service* atau *IP virtual* yang didefinisikan sebelumnya saat menjalankan *controller.* Selanjutnya paket akan dikirim ke tujuan melalui *switch.*

3. *Controller* melakukan pengecekan paket dari *client*

Seperti yang dijelaskan pada tabel 3.5, *switch* kembali meminta keputusan pada *controller* untuk memberikan aksi pada paket tersebut. Pertama kali dilakukan *parsing header* paket untuk menentukan jenis paket yang masuk. Kemudian dilakukan pengecekan didalam *flow table.* Jika ada, maka akan dilakukan aksi pada paket tersebut. Jika tidak, maka akan dilakukan pengecekan kembali apakah paket tersebut *TCP* dan *Ipv4.* Jika iya, maka akan dilakukan penyimpanan pada *flow table* dan akan dilakukan *load balancing.*

Pada *source code* ini menjelaskan proses perhitungan penentuan *server* dari metode *round robin*

Tabel 3. 6 *Source code* Penentuan *Server* Metode *Round Robin*

Source Code Metode *Round Robin*	
1	`def _pick_server (self, key, inport):`
2	`    self.last_server =  (self.last_server + 1) %`
3	`len(self.live_servers)`
4	`    return self.live_servers.keys()[self.last_server]`

Tabel 3.6 merupakan kode penetuan server metode *round robin* yang dapat dijelaskan sebagai berikut :

- Baris 1 adalah nama method dari metode *round robin.*
- Baris ke 2 sampai 3 adalah mengambil *server* terakhir yang melayani *request* yang selanjutnya dilakukan penambahan nilai 1, dan dilakukan proses *modulo* dengan nilai total semua *server*.
- Baris ke 4 adalah mengambil *IP server* dari *server* yang terpilih dari *server* yang tersedia.

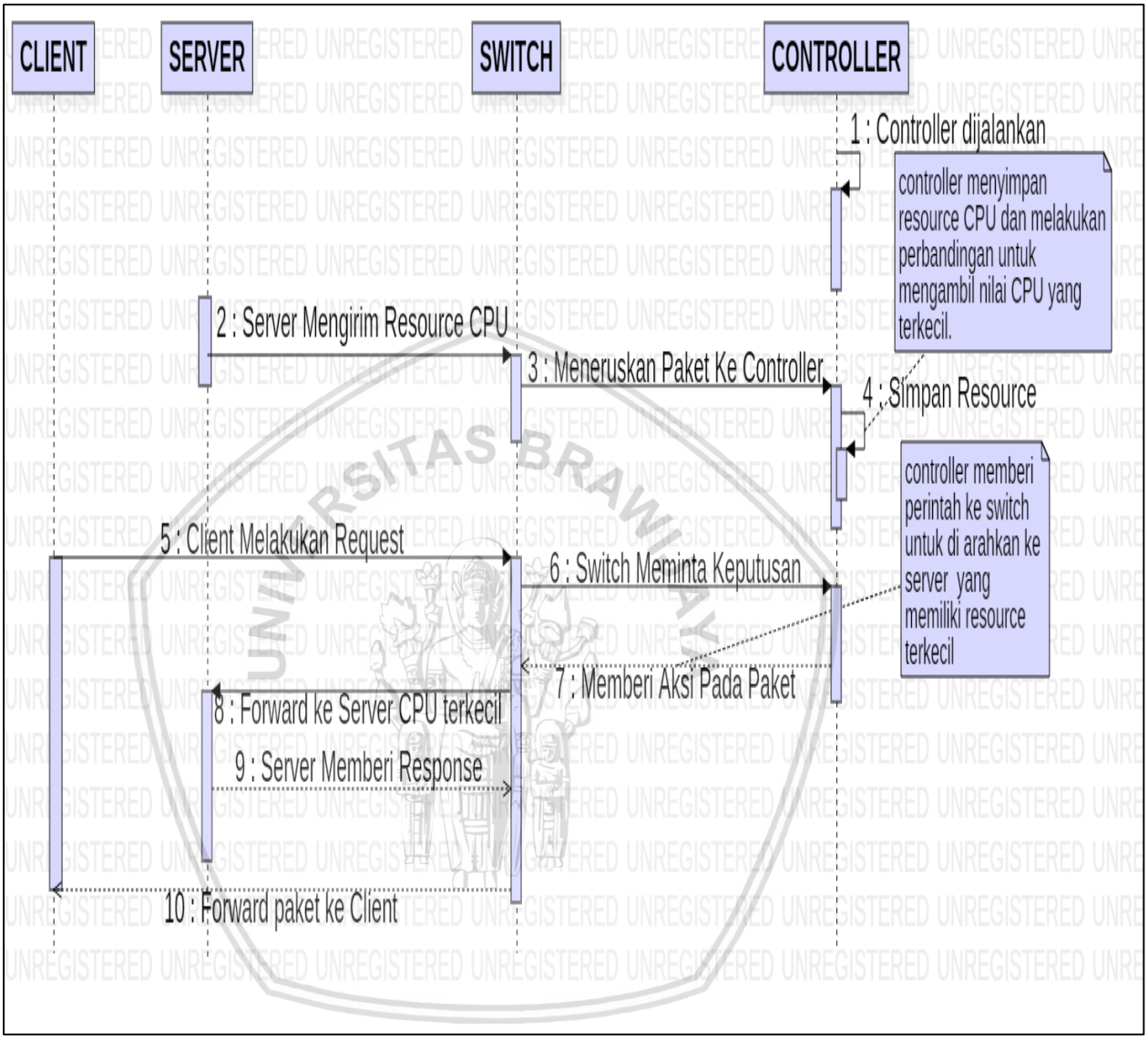
4. *Switch* meneruskan paket ke *server* yang sudah ditentukan.

5. *Server* memberikan *respon* ke *client*

Setelah *server* menerima *request client,* maka *server* tersebut akan memberikan respon dan mengirim paket balasan ke *switch*. Jika paket tersebut dengan *header*nya telah ada pada *tabel flow* maka dilakukan aksi terhadap paket tersebut. Jika tidak, maka *switch* meminta kembali aksi terhadap *controller* dan melakukan *update* terhadap *table openflow*.

3.4.2.5 Metode Berbasis *CPU Usage*

Gambar 3.12 menjelaskan bagaimana proses aliran data dari metode berbasis *CPU usage.*



Gambar 3. 12 *Data Flow* Metode Berbasis *CPU Usage*

Gambar 3.12 merupakan gambar *data flow metode berbasis CPU usage* secara keseluruhan. Proses dibagi menjadi beberapa langkah, yaitu :

1. Menjalankan *Controller*

Pertama yang harus dilakukan adalah menjalankan *controller.* Dalam menjalankan *controller* harus menjalankan *file* yang berisi metode berbasis *CPU usage*, kemudian mendefinisikan *IP virtual* yang diakses *client* dan *server*. *Controller* melakukan proses penjadwalan terhadap *resource* yang didapatkan dari *server*, yaitu dengan memilih *server* yang memiliki kondisi *CPU* terkecil. *Controller* juga harus dapat mendeteksi sebuah *server* dalam kondisi hidup atau tidak dengan mengirimkan paket *ARP*. *Controller* akan melakukan pengecekan kondisi *up* atau

down dengan melakukan perhitungan seperti yang dijelaskan pada tabel 3.3 dan tabel 3.4 yang didapatkan dari pengiriman paket *ARP* ke *server* sampai *server* memberi balasan dalam bentuk *ARP-REPLAY* ke *controller*.

2. *Server* mengirimkan *resource CPU*

Ketika *server* sudah dalam kodisi *up, server* siap mengirimkan paket yang berisi penggunaan *CPU* menggunakan protokol *UDP.* Pada *source code* berikut ini menjelaskan proses pengiriman *resource server*.

Tabel 3. 7 *Surce Code* Pengiriman *Resource CPU*

Source Code* Pengiriman *Resource CPU	
1	`import socket`
2	`import psutil`
3	`import pickle`
4	`def getresource():`
5	`    cpu_usage = psutil.cpu_percent(interval=1)`
6	`    return pickle.dumps(cpu_usage)`
7	`server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`
8	`server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)`
9	`while True:`
10	`    server.connect(('192.168.1.100', 5000))`
11	`    server.send(getresource())`
12	`    time.sleep(1)`

Tabel 3.7 merupakan kode pengiriman *resource CPU* yang dapat dijelaskan sebagai berikut.

- baris ke-1,ke-2 dan ke-3 menjelaskan *import* modul dari *psutil* untuk mengambil *resource CPU* dari *server*.
- baris ke-4 sebuah method yang berfungsi untuk mengambil *resource.*
- baris ke-5 menjelaskan pengambilan *resource* pengguna *CPU* dengan satuan % dalam interval 1 detik.
- baris ke-6 menjelaskan pengembalian nilai *CPU* dengan membungkusnya ke dalam representasi data berupa *pickle*.
- baris ke 7-8 melakukan pembuatan koneksi *UDP*
- baris ke 9-12 merupakan pembuatan koneksi dengan *IP* virtual dan *port* 5000. dan memanggil *method resource* untuk dikirim dengan interval setiap 1 detik.

3. *Switch* meneruskan paket dari ke tujuan yaitu *controller*

4. *Controller* melakukan pengecekan *server* paket *resource CPU* dan melakukan pemilihan *server* yang *CPU* yang terkecil

Ketika *server* mengirimkan paket ke *controller* melalui *switch*, akan dilakukan pengecekan paket terlebih dahulu untuk memberikan aksi pada paket tersebut. Jika paket berupa *UDP* dan paket tersebut dari *server,* maka *controller* akan menyimpan paket tersebut. Jika tidak paket tersebut akan di *drop*. Pada proses yang lain *controller* memiliki tugas melakukan perhitungan untuk memilih

server yang memiliki *CPU* terkecil untuk digunakan proses *load balancing*. Berikut *source code* untuk pengecekan paket *resource CPU.*

Tabel 3. 8 *Source Code* Pengecekan *Resource CPU*

Source Code Pengiriman ***Resource CPU***	
1	`if udpp and ipp:`
2	`    if ipp.dstip == self.service_ip and ipp.srcip in`
3	`self.live_servers`
4	`        self.resources[ipp.srcip] = float(pickle.loads(udpp.next))`
5	`        log.debug(self.resources)`
6	`        return drop()`

Tabel 3.8 merupakan kode pengecekan *resource CPU* yang dapat dijelaskan sebagai berikut.

- Baris 1 dilakukan pengecekan apakah paket yang datang berupa *UDP.* Jika iya, akan dilakukan *statement* berikutnya.
- Baris 2-4 jika *IP* tujuan paket tersebut sama dengan *IP service* atau *IP* virtual dan *IP* sumber sama dengan *IP server* yang hidup, maka *resource* akan disimpan.
- Baris ke 5 menjelaskan bahwa *resource* akan dicetak di *controller.*
- Baris ke 6 jika paket yang datang bukan *UDP,* maka paket akan di *drop.*

Tabel 3. 9 *Source Code* Pemilihan *Server* dengan *CPU Usage* Terkecil

Source Code Pemilihan *Server* dengan *CPU usage* terkecil	
1	`def resource_algorithm(self):`
2	`    while RUNNING is True:`
3	`        if len(self.resources) == 0:`
4	`            self.cpu_usage_terkecil = self.servers[0]`
5	`        server_resource = min(self.resources,`
6	`key=self.resources.get)`
7	`        self.cpu_usage_terkecil = server_resource`
8	`        time.sleep(0.5)`

Tabel 3.9 merupakan kode pemilihan *server* dengan *CPU* terkecil yang dapat dijelaskan sebagai berikut.

- Baris 1 adalah nama *method* yang dipakai.
- Baris ke 2 melakukan perulangan secara terus menerus.
- Baris ke 3 melakukan pengecekan jika nilai *resource* tidak ada, maka nilai *resource* terkecil akan di atur ke *server* dengan index 0.
- Baris ke 5 sampai 7 melakukan perhitungan *resource* terkecil dan menyimpannya.
- Baris ke 8 melakukan perulangan dengan waktu *time sleep* 0,5 detik.

5. *Client* melakukan *request* dengan mengakses *IP service*

Client melakukan *request* menggunakan *httperf* dengan mengakses *IP service* atau *IP virtual* yang didefinisikan sebelumnya saat menjalankan *controller.* Selanjutnya paket akan dikirim ke tujuan melalui *switch.*

6. *Switch* meneruskan paket dari server ke tujuan yaitu *controller*

7. *Controller* melakukan pengecekan paket dari *client*

Seperti yang dijelaskan pada tabel 3.5, *Switch* kembali meminta keputusan pada *controller* untuk memberikan aksi pada paket yang masuk. Pertama kali yang harus dilakukan adalah melakukan *parsing header* pada paket tersebut. Kemudian dilakukan pengecekan didalam *flow table.* Jika ada, maka akan dilakukan aksi pada paket tersebut. Jika tidak, maka akan dilakukan pengecekan kembali apakah paket tersebut *TCP* dan *Ipv4.* Jika iya, maka akan dilakukan penyimpanan pada *flow table* dan akan dilakukan *load balancing*.

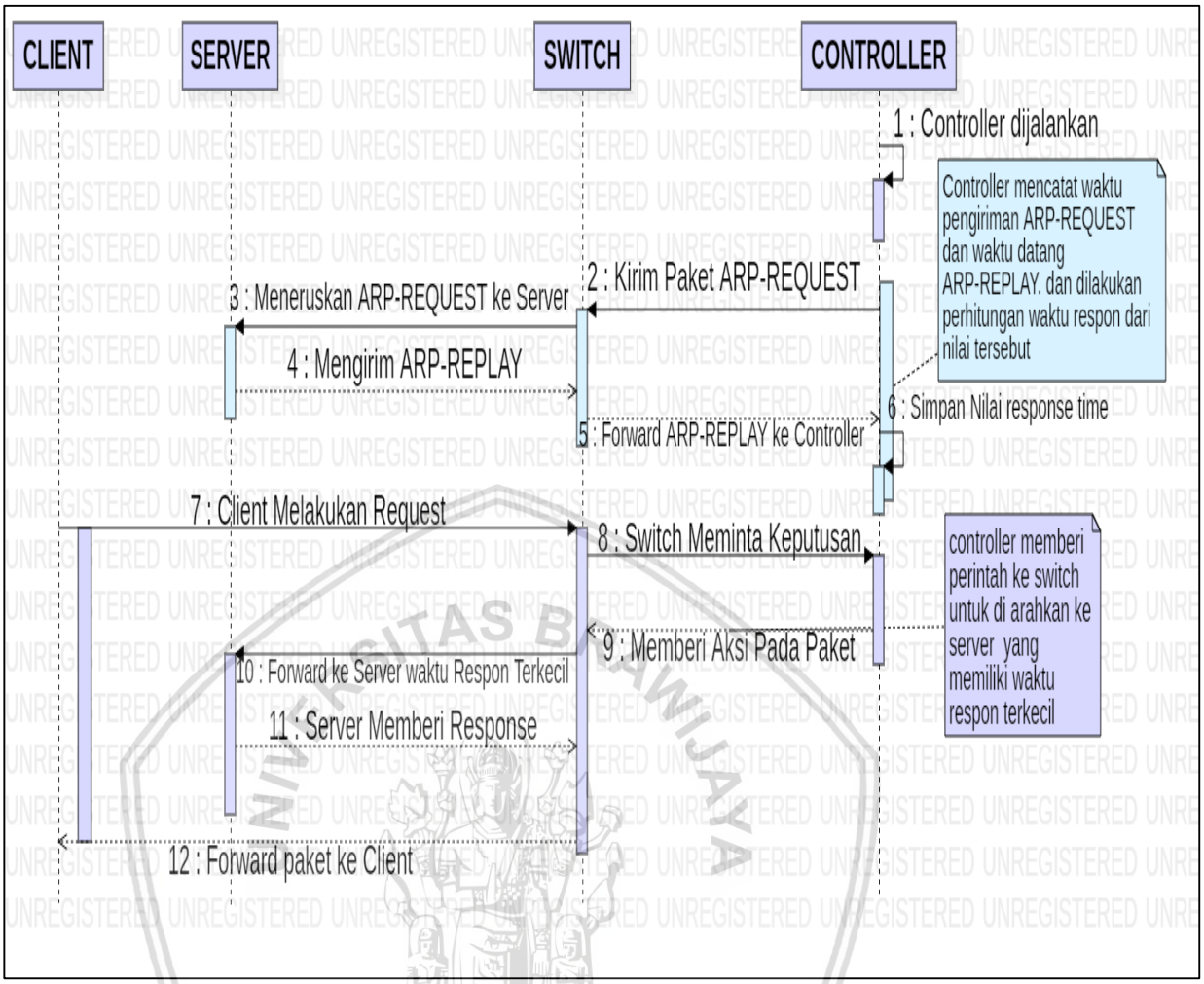
8. *Switch* meneruskan paket ke *server* yang memiliki penggunaan *CPU* terkecil
.

9. *Server* memberikan *respon* ke *client*

Setelah *server* menerima *request* dari *client,* kemudian *server* tersebut akan memberikan respon dan mengirim paket balasan ke *switch*. Jika paket tersebut dengan *header*nya telah ada pada *tabel flow,* maka dilakukan aksi terhadap paket tersebut. Jika tidak maka *switch* meminta kembali aksi terhadap paket tersebut pada *controller* dan melakukan *update* terhadap *flow table*.

3.4.2.6 Metode Berbasis *Response Time*

Pada Gambar 3.13 menjelaskan bagaimana proses aliran data dari metode *berbasis response time.*



Gambar 3. 13 *Data Flow* Metode Berbasis *Response Time*

Gambar 3.13 merupakan gambar *data flow metode berbasis response time* secara keseluruhan. Proses dibagi menjadi beberapa langkah, yaitu:

1. Menjalankan *Controller*

Pertama yang harus dilakukan adalah menjalankan *controller. Controller* bertugas melakukan proses penjadwalan dengan mengarahkan *traffic* ke *server* yang mempunyai nilai *server* terkecil. *Controller* juga harus dapat mendeteksi sebuah *server* dalam kondisi hidup atau tidak dengan mengirimkan paket *ARP*. *Controller* akan melakukan pengecekan kondisi *up* atau *down* dengan melakukan perhitungan seperti yang dijelaskan pada tabel 3.3 dan tabel 3.4 yang didapatkan dari pengiriman paket *ARP* ke *server* sampai *server* memberi balasan dalam bentuk *ARP-REPLAY* ke *controller*.

2. Pada proses no 2 sampai no 6 merupakan proses untuk mendapatkan waktu respon dari *server*. Proses yang dilakukan sama seperti proses pengiriman *ARP* untuk melakukan pengecekan status *server.* Ada beberapa penambahan variabel seperti variabel yang digunakan untuk menyimpan waktu respon dari *server* dan variabel yang mempunyai nilai 999 yang digunakan sebagai nilai *default* jika *server* dalam kondisi *down* .

Tabel 3. 10 *Source Code* Pengiriman *ARP-REQUEST* untuk Mendapatkan *Response Time*

Source Code Pengiriman *ARP-REQUEST* untuk Mendapatkan *Response Time*

```
1     def _do_probe (self):
2       self.server_response_time = {}
3       self._do_expire()
4       server = self.servers.pop(0)
5       self.servers.append(server)
6
7       r = arp()
8       r.hwtype = r.HW_TYPE_ETHERNET
9       r.prototype = r.PROTO_TYPE_IP
10      r.opcode = r.REQUEST
11      r.hwdst = ETHER_BROADCAST
12      r.protodst = server
13      r.hwsrc = self.mac
14      r.protosrc = self.service_ip
15      e = ethernet(type=ethernet.ARP_TYPE, src=self.mac,
16                   dst=ETHER_BROADCAST)
17      e.set_payload(r)
18      #self.log.debug("ARPing for %s", server)
19      msg = of.ofp_packet_out()
20      msg.data = e.pack()
21      msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
22      msg.in_port = of.OFPP_NONE
23      self.con.send(msg)
24
25      self.outstanding_probes[server] = time.time() +
26  self.arp_timeout
        self.server_response_time[server] = 999
```

Tabel 3.10 merupakan kode pengiriman *ARP-REQUEST* untuk mendapatkan *response time* yang dapat dijelaskan sebagai berikut :

- Baris 1 merupakan nama *method* untuk mengirim *ARP-REQUEST*
- Baris 2 merupakan variabel untuk menyimpan *response time* dari *server* yang aktif
- Baris ke 3 merupakan *method* untuk melakukan pengecekan apakah *server time out* atau tidak.
- Baris 4 sampai 5 merupakan proses mengambil *IP server* untuk dilakukan proses pengiriman *ARP-REQUEST* . Pengiriman dilakukan kepada *server* secara bergantian.
- Baris 7 sampai 23 merupakan konstruksi pengiriman *ARP* antara lain : Jenis paket yang dikirim *ARP-REQUEST,* jenis pengiriman *broadcast,* tujuan ke *server*, *mac address, IP server,* jenis paket *out,* dsb.
- Baris 25 merupakan inisialisai variabel untuk menyimpan waktu kirim *probes* ke setiap *server* ditambah dengan waktu *time out.*
- Baris 26 merupakan nilai *default* yang digunakan ketika server mengalami *down* atau *time out.*

3. *Switch* meneruskan paket ke *server*

4. Ketika server menerima *ARP-REQUEST,* server akan memberi balasan berupa *ARP-REPLAY* pada *IP* sumber yaitu *controller* .

5. *Switch* akan meneruskan paket *ARP-REPLAY* ke *IP* sumber yaitu *controller.*

6. *Server* memberi balasan berupa *ARP-REPLAY.* Batas waktu *arp_time_out* adalah 3 detik. Jika *server* dalam waktu 3 detik tidak memberikan balasan berupa *ARP-RELAY,* maka *server* dinyatakan *down* dan nilai *response time* akan di atur pada nilai *default* yaitu 999. Proses perhitungan dilakukan dengan membandingkan waktu sekarang apakah lebih besar dibandingkan dengan waktu *expire* (waktu kirim *ARP_REQUEST + time_out*). Jika waktu sekarang lebih besar dari pada waktu *expire*, maka *server* dinyatakan *time out*. Berikut *source code* perhitungan waktu *response time* dari *server* dan *source code* pemilihan *response time* terkecil.

Tabel 3. 11 *Source code* Pengecekan *ARP-REPLAY* dan Perhitungan *Response Time*

	Source Code* Pengecekan Paket *ARP-REPLAY* dan Perhitungan *Response Time
1	`def _do_expire (self):`
2	
3	`t = time.time()`
4	
5	`for ip,expire_at in self.outstanding_probes.items():`
6	`  if t > expire_at:`
7	`  self.outstanding_probes.pop(ip, None)`
8	`  if ip in self.live_servers:`
9	`  self.log.warn("Server %s down", ip)`
10	`  del self.live_servers[ip]`
11	`  del self.server_respose_time[ip]`
12	
13	`if not tcpp:`
14	`  arpp = packet.find('arp')`
15	`  if arpp:`
16	`  if arpp.opcode == arpp.REPLY:`
17	`  if arpp.protosrc in self.outstanding_probes:`
18	`  self.server_response_time[arpp.protosrc]`
19	`= arrive_time-(self.outstanding_probes[arpp.protosrc]-`
20	`self.arp_timeout)`
21	`  del self.outstanding_probes[arpp.protosrc]`
22	`  if (self.live_servers.get(arpp.protosrc, (None,None))`
23	`  == (arpp.hwsrc,inport)):`
24	`  pass`
25	`  else:`
26	`  self.live_servers[arpp.protosrc] = arpp.hwsrc,inport`
27	`  self.log.info("Server %s up", arpp.protosrc)`
	`  return`

Tabel 3.11 merupakan kode pengecekan *ARP-REQUEST* yang dapat dijelaskan sebagai berikut :

- Baris 1 merupakan nama *method* yang digunakan untuk melihat apakah *server* dalam keadaan *time out*
- Baris 3 merupakan inisialisasi bahwa t = waktu sekarang

- Baris 5 sampai 9 menjelaskan proses perbandingan yang digunakan untuk menentukan apakah *server* dalam kondisi *down* apa tidak, dengan melakukan perbandingan apakah waktu sekarang apakah lebih besar dari waktu *expire.* Jika lebih besar, *server* akan dinyatakan dalam kondisi *down.*
- Baris 10 merupakan perintah untuk menghapus *server* yang dinyatakan dalam kondisi *down*
- Baris 11 merupakan merupakan perintah untuk menghapus *response time* dari server yang dinyatakan *down*.
- Baris 13 sampai 16 merupakan jika paket bukan *TCP* melainkan berupa *ARP-REPLAY* dilakukan *statement* didalamnya.
- Baris 17 sampai 19 menjelaskan proses perhitungan nilai *response time* dari *server,* dengan mencatat waktu datangnya paket dikurangi waktu pengiriman. Hasilnya adalah *response time* yang kemudian di simpan.
- Baris 20 merupakan jika *server* memberi balasan, maka *outstanding_probes* akan dihapus.
- Baris 25 sampai 27 merupakan *update* kodisi *server* dan *IP* nya pada *dictionary live_server* dan menampilkan *server* dalam kondisi *up* pada *controller*.

Tabel 3. 12 *Source code* Memilih *Server* Berdasarkan *Response Time* Terkecil

***Source Code* Memilih *Server* Berdasarkan *Response Time* Terkecil**	
1	`def _do_algorithm(self):`
2	`    while RUNNING is True:`
3	`        if len(self.server_response_time) == 0:`
4	`            self.response_time_terkecil =`
5	`self.live_servers.keys()[0]`
6	`        server_ip = min(self.server_response_time,`
7	`key=self.server_response_time.get)`
8	`        self.response_time_terkecil = server_ip`
9	`        time.sleep(0.5)`
10	

Tabel 3.12 merupakan kode memilih *server* berdasarkan *response time* terkecil yang dapat dijelaskan sebagai berikut :

- Baris ke 1 nama *method* yang digunakan untuk memilih server berdasarkan *response time* terkecil.
- Baris ke 2 melakukan perulangan secara terus menerus.
- Baris ke 3-5 melakukan pengecekan apabila nilai *response time* dari *server* tidak ada, maka nilai respon terkecil di atur pada *server* dengan index 0.
- Baris 6-8 melakukan pemilihan nilai respon terkecil dari *server* dan menyimpannya
- Baris ke 9 melakukan perulangan dengan waktu time sleep 0,5 detik

7. *Client* melakukan *request* dengan mengakses *IP service*

Client melakukan *request* menggunakan httperf dengan mengakses IP service atau *IP virtual* yang didefinisikan sebelumnya saat menjalankan *controller.* Selanjutnya paket akan dikirim ke tujuan melalui *switch.*

8. *Switch* meneruskan paket dari server ke tujuan yaitu *controller*

9. *Controller* melakukan pengecekan paket dari *client*

Seperti yang dijelaskan pada tabel 3.5, *Switch* kembali meminta keputusan pada *controller* untuk memberikan aksi pada paket tersebut. Pertama kali dilakukan *parsing header* paket untuk menentukan jenis paket yang masuk. Kemudian dilakukan didalam *flow table.* Jika ada, maka akan dilakukan aksi pada paket tersebut. Jika tidak, maka akan dilakukan pengecekan kembali apakah paket tersebut *TCP* dan *Ipv4.* Jika iya, maka akan dilakukan penyimpanan pada *flow table* dan akan dilakukan *load balancing*.

10. *Switch* meneruskan paket ke *server* yang memiliki *response time* terkecil yang sudah dilakukan perhitungan sebelumnya.

11. Server memberi *response* kepada *client.*

3.5 Pengujian dan Analisis

Pengujian dan analisis dilakukan setelah implementasi sistem berhasil dilakukan. Pengujian dan analisis dilakukan untuk mengetahui apakah penelitian telah dijalankan sesuai dengan skenario yang telah dirancang.

3.5.1 Tujuan Pengujian

Tujuan pengujian ini adalah untuk mengetahui perbandingan kinerja dari metode *round robin*, metode berbasis *CPU usage* dan metode berbasis *response time* untuk *load balancing* pada *sofware defined network* . Pengujian juga dilakukan untuk mengetahui hasil dari nilai rata-rata setiap paramater yang diberikan. *Request* digunakan untuk memberikan beban permintaan *client* kepada *server* dalam satu waktu sehingga dapat diketahui kemampuan *server* dalam melakukan respon terhadap *client*. *Rate* pada pengujian ini digunakan untuk memberikan ukuran kecepatan *bit* data transmisi dalam satu waktu. Selanjutnya hasil dari masing-masing percobaan akan dibandingkan dan dilakukan analisis dari hasil yang didapatkan. Kemudian dapat diketahui perbandingan kinerja dari metode *round robin*, metode berbasis *CPU usage* dan metode berbasis *response time*.

3.5.2 Proses Pengujian

Proses pengujian diawali dengan memberikan berapa banyak koneksi dengan *rate* yang berbeda. Hal ini dijalankan menggunakan *tools httperf*. Perintah untuk menjalankan proses ini adalah:

```
httperf –server [ip service] --port 80 --num-conns a --rate b
```

Pada *tool httperf* dimulai dengan memasukkan *IP service* yaitu 192.168.2.100, lalu membuka *port* untuk diakses *client* yaitu *port* 80, *-num-conns*

adalah untuk memberikan berapa banyak koneksi dalam 1 waktu sedangkan *rate* adalah besa kecepatan *bit* data dalam 1 waktu. Dengan a dan b adalah bilangan bulat positif.

3.5.3 Skenario Pengujian

Pengujian menggunakan *httperf* menggunakan parameter *throughput* dengan satuan *KB/s, response time* dengan satuan *ms,* dan *CPU usage* dengan satuan persentase (%). Nilai pengujian *throughput* dan *response time* diambil dari percobaan yang dilakukan sebanyak 5 kali. Sedangkan pengujian *CPU Usage* hanya dilakukan 1 kali, karena sudah mencakup nilai variasi *CPU Usage*. Pengujian dilakukan pada masing-masing metode dengan kondisi yang sama.

Skenario pengujian peratama kali dilakukan dengan mencari besar *rate* maksimal yang mampu ditangani sistem. Pada proses mencari maksimal *rate* dilakukan *request* menggunakan *httperf* dimula dengan *rate* dengan nilai kecil dibawah 100 hingga ditemukan maksimal *rate*. Nilai *rate* maksimal yang didapatkan yaitu 360 *req/s.* Selanjutnya nilai *rate* dikategorikan menjadi 3 kategori yaitu *low* , *medium* , dan *high*. Nilai *high* adalah 360 *req/s,* untuk nilai *medium* adalah setengah dari nilai *high* yaitu 180 *req/s*, dan untuk nilai *low* adalah setengah dari nilai *medium* yaitu 90 *req/s*. Pengujian dari masing-masing *rate* dilakukan dalam waktu 10 detik, kemudian didapatkan banyak koneksi untuk skenario ini yaitu :

- *Low* : 90 *req/s* (*rate*) dengan jumlah koneksi 900.
- *Medium* : 180 *req/s* (*rate*) dengan jumlah koneksi 1800.
- *High*: 360 *req/s* (*rate*) dengan jumflah konesi 3600.

Skenario pengujian *throughput, response time* dan *CPU usage* dilakukan dengan kondisi setiap *server* menjalankan mejalankan vidio yang sama untuk memberikan beban pada masing-masing *server*. *Client* melakukan *request* menggunakan *httperf* sesuai kategori *rate*. Setelah dilakukan *request* dari masing-masing kategori *rate*, maka didapatkan nilai *throughput* dan *response time.* Selanjutnya diambil rata–rata dari 5 kali pengujian.

3.6 Kesimpulan dan Saran

Pengambilan kesimpulan dilakukan setelah semua tahapan perancangan, implemetasi, dan pengujian sistem telah selesai dilakukan. Kesimpulan diambil dari hasil pengujian dan analisis terhadap penelitian yang telah dilakukan. Tahap terakhir penulisan adalah saran yang dimaksudkan untuk memberikan pertimbangan atas pengembangan penelitan yang lebih lanjut.

BAB 4 HASIL

Pada bab ini ditunjukan seluruh hasil pengujian yang dilakukan dari metode *round robin,* metode berbasis *response time* dan metode berbasis *CPU usage.* Hasil pengujian menampilkan nilai dari *throughput, response time,* dan *CPU usage.*

4.1 Hasil Pengujian

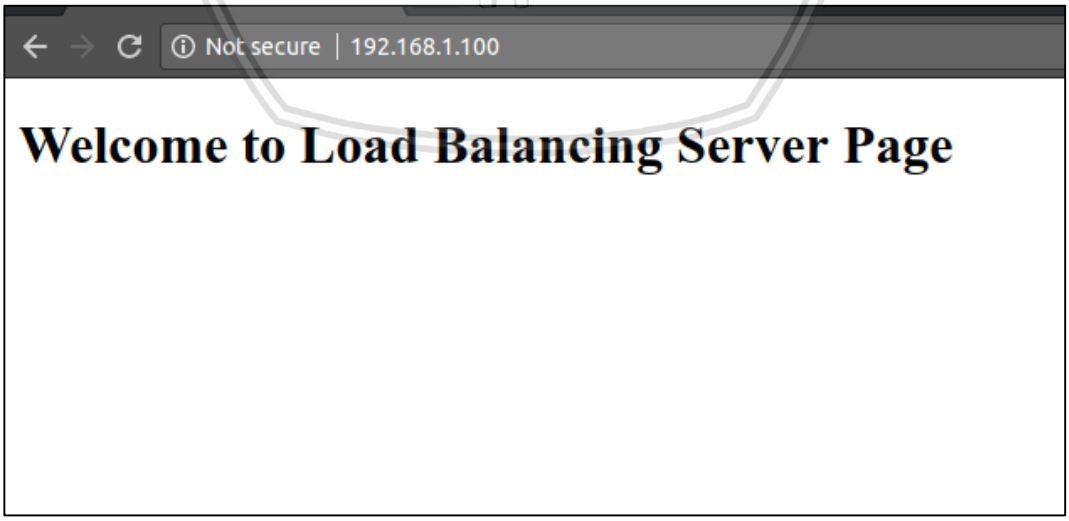
Pengukuran parameter *throughput, response time, dan CPU time* dengan menggunakan *tools httperf* pada jaringan rancangan berdasarkan topologi jaringan yang telah dibangun.

4.1.1 Pengujian Perancangan Sistem

Pada hasil pengujian menunujukan bahwa perancangan telah berhasil dijalankan sesuai dengan perancangan pada bab sebelumnya. Pada pengujian perancangan dilakukan *request* dari *client* dengan melakukan "curl" dari terminal maupun melakukan akses dari *web* browser. Alamat yang diakses adalah *IP service* dari sistem yaitu 192.168.1.100. Pengujian perancangan dilakukan hingga sistem benar-benar bisa berjalan sesuai dengan rencana yang diinginkan. Sehingga dapat dilakukan pengujian kinerja untuk masing – masing metode.

4.1.1.1 *Server*

Sebelum client melakukan request ke *server*, setiap *server* harus memiliiki sebuah *web server* agar *server* dapat merespon *request* yang dilakukan oleh *client*. *Web server* yang digunakan adalah *flask web server.* Berikut gambar dari web *server* tersebut.

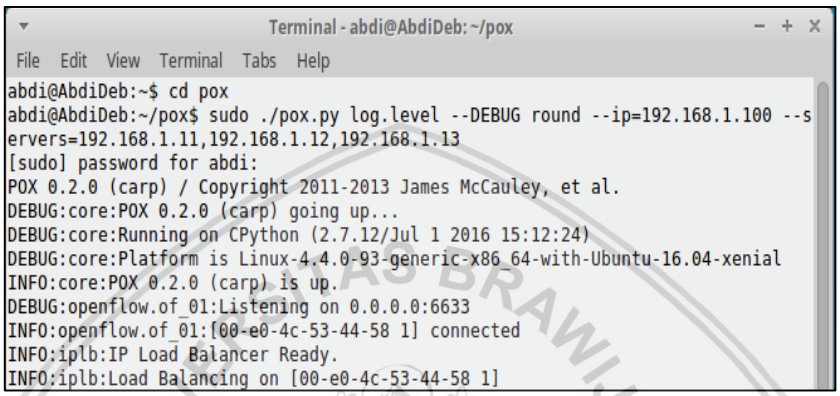


Gambar 4. 1 *Web Server Flask* Sudah Berjalan pada Setiap *Server*

Pada gambar 4.1 menunjukkan bahwa *webserver flask* dapat diakses dari komputer *client*. Hal tersebut membuktikan bahwa setiap *server* telah dalam keadaan up dan dapat bejalan dengan baik.

4.1.1.2 SDN Switch

Switch yang digunakan adalah *Open vSwitch* (*switch virtual*) yang diimplementasikan pada sebuah komputer dekstop. Jenis *Open vSwitch* yang digunakan yaitu *Open vSwitch 2.2.5*.

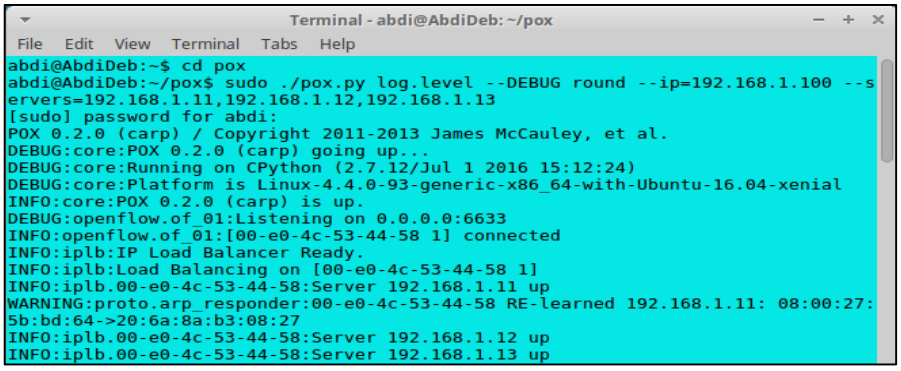


Gambar 4. 2 *SDN Switch*

Pada gambar 4.2 menunjukkan bahwa *switch* telah terpasang protokol *openflow* yang dibuktikan dengan mampunya *switch* mampu melakukan *forwarding* yang ditampilkan pada *controller*. Terlihat juga pada terminal "INFO : *openflow* .of_01 : [Mac Address] connected".

4.1.1.3 Controller

Controller bertugas mengelola kontrol aliran ke *switch* / *router* (melalui *API*) dan mengatur jalur mana yang akan dipilih. *Controller* yang digunkan adalah *POX*. *Controller* akan menjalankan metode *loadbalancing* dan melakukan pengecekan *server* terlebih dahulu, apakah *server* dalam keadaan hidup atau tidak.

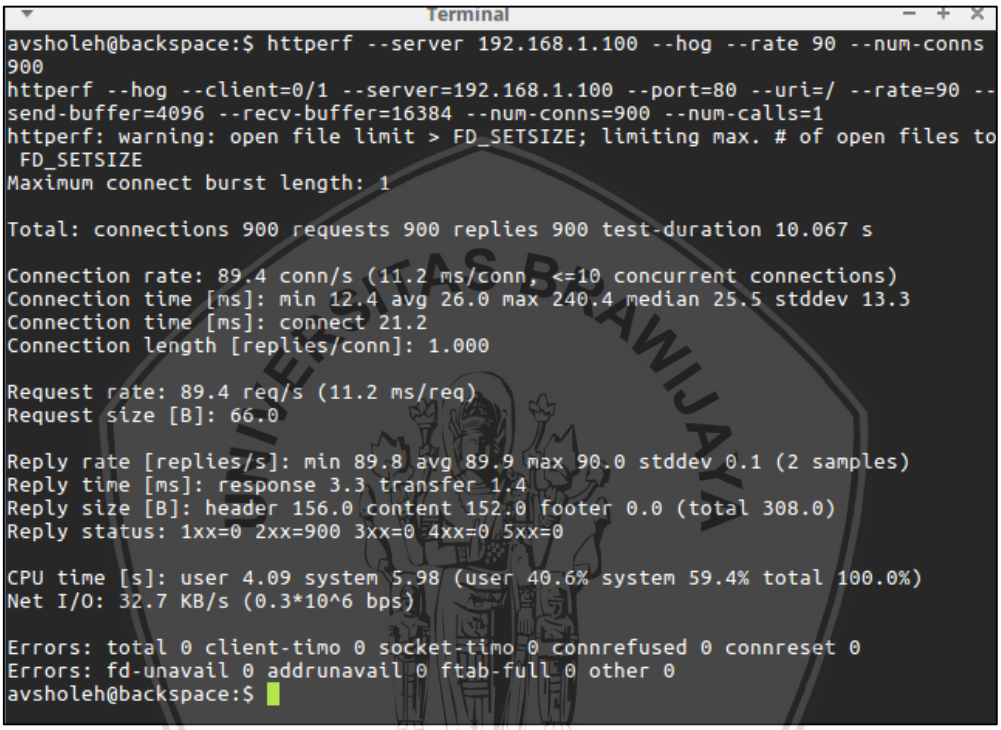


Gambar 4. 3 Tampilan *Controller POX*

Pada gambar 4.3 menunjukkan *controller POX* telah berhasil terpasang dan terhubung dengan *switch* dan ketiga *server virtual* yang ada dengan kondisi *server* dalam keadan up.

4.1.1.4 Client

Untuk melakukukan pengujian, *client* harus terpasang *tools httperf* terlebih dahulu. Dalam pengujian ini *tools httperf* digunakanan untuk mendapatkan nilai *throughput* dan *response time.* Berikut tampilan dari *client* waktu melakukan pengujian menggunakan *httperf*.



Gambar 4.4 Tampilan *Client* Melakukan *Request*

Pada gambar 4.4 menunjukkan bahwa *client* sudah terpasang *tools httperf* dan mampu melakukan *request* kepada *server*. Sehingga parameter yang diperlukan mampu ditampilkan pada *tools httperf* setelah seluruh *request* mendapat respon dari *server*. Nilai *throughput* ditampilkan pada *Net I/O* dan *response time* ditampilkan pada *replay time.*

4.1.2 Pengujian *Throughput*

Proses pengujian *throughput* dapat dilihat dari nilai perbandingan *throughput* antara metode *round robin,* metode berbasis *response time* dan metode berbasis *CPU usage* . Satuan yang digunakan untuk *throughput* yaitu KB/s. *Throughput* yang diamati didapatkan dari hasil pengujian dari semua kategori *rate* yaitu: *low ,medium* , dan *high* dengan 5 kali percobaan. Nilai tersebut merupakan jumlah total *transfer* data yang sukses dilakukan. Berikut hasil yang diperoleh :

Tabel 4. 1 Hasil Pengujian *Throughput* Kategori *Rate Low* (90)

No	LOW		
	Round Robin (KB/s)	Response Time Based (KB/s)	CPU Usage Based (KB/s)
1	32,7	32,6	31,7
2	32,7	32,6	31,2
3	32,7	30,9	29,9
4	32,7	30,0	29,9
5	32,4	28,6	28,3
Rata-rata	32,64 (KB/s)	30,86 (KB/s)	30,20 (KB/s)

- Pada jumlah rate *low* pengujian *throughput* dengan metode *round robin* didapatkan nilai dengan rata-rata 32,64 KB/s, pada metode berbasis *response time* didapatkan nilai *throughput* dengan rata-rata 30,86 KB/s, sedangkan pada metode berbasis *CPU Usage* pengujian *throughput* nilai yang didapatkan nilai throughput dengan rata-rata adalah 30,20 KB/s.

Tabel 4. 2 Hasil Pengujian *Throughput* Kategori *Rate Medium* (180)

No	MEDIUM		
	Round Robin (KB/s)	Response Time Based (KB/s)	CPU Usage Based (KB/s)
1	61,6	65,9	65,4
2	60,6	65,3	65,3
3	59,9	65,3	64,2
4	59,8	65,3	60,3
5	59,7	63,3	60,1
rata-rata	60,32 (KB/s)	65,02 (KB/s)	63,06 (KB/s)

- Pada jumlah rate medium pengujian *throughput* dengan metode *round robin* didapatkan nilai dengan rata-rata 60,32 KB/s, pada metode berbasis *response time* didapatkan nilai *throughput* dengan rata-rata 65,02 KB/s, sedangkan pada metode berbasis *CPU Usage* pengujian *throughput* nilai yang didapatkan nilai dengan rata-rata 63,06 KB/s.

Tabel 4. 3 Hasil Pengujian *Throughput* Kategori *Rate High* (360)

No	HIGH		
	Round Robin (KB/s)	Response Time Based (KB/s)	CPU Usage Based (KB/s)
1	105,4	119,5	118,2
2	103,5	118,7	117,4
3	94,0	117,7	113,4
4	93,0	112,0	113,3
5	84,0	110,1	112,1
Rata-rata	96,10 (KB/s)	115,50 (KB/s)	114,88 (KB/s)

- Pada jumlah rate *high* pengujian *throughput* dengan metode *round robin* didapatkan nilai dengan rata-rata 96,10 KB/s, pada metode berbasis *response time* didapatkan nilai *throughput* dengan rata-rata 115,5 KB/s,sedangkan pada metode berbasis *CPU Usage* pengujian *throughput* didapatkan nilai dengan rata-rata 114,88 KB/s.

4.1.3 Pengujian *Response Time*

Proses pengujian *responsse time* dapat dilihat dari nilai perbandingan *response time* antara metode *round robin,* metode berbasis *response time* dan metode berbasis *CPU usage*. Satuan yang digunakan untuk hasil pengujian *response time* yaitu *ms*. *Response time* yang diamati didapatkan dari hasil pengujian dari semua kategori *rate* yaitu: *low ,medium* , dan *high* dengan 5 kali percobaan. Berikut hasil yang diperoleh :

Tabel 4. 4 Hasil Pengujian *Response Time* Kategori *Rate Low* (90)

No	LOW		
	Round Robin (ms)	Response Time Based (ms)	CPU Usage Based (ms)
1	26,2	24,2	44,3
2	28,3	24,8	43,3
3	30,5	26,5	43,1
4	30,9	28,3	42,4
5	35,7	28,7	40,6
Rata-rata	30,32 (ms)	26,50 (ms)	42,76 (ms)

- Pada pengujian rate *low* pengujian *response time* dengan metode *round robin* memiliki nilai rata-rata 30,32 ms, pada metode berbasis *response time* didapatkan nilai *response time* dengan rata-rata 26,50 ms, sedangkan pada metode berbasis *CPU Usage* pengujian *response time* nilai dengan rata-rata 42,76 ms.

Tabel 4. 5 Hasil Pengujian *Response Time* Kategori *Rate Medium* (180)

No	MEDIUM		
	Round Robin (ms)	Response Time Based (ms)	CPU Usage Based (ms)
1	81,3	65,6	77,6
2	91,6	65,9	88,8
3	106,6	68,4	88,5
4	107,8	68,7	90,6
5	126,2	69,2	89,2
Rata-rata	102,70 (ms)	67,56 (ms)	86,94 (ms)

- Pada pengujian jumlah rate medium pengujian *response time* dengan metode *round robin* didapatkan nilai rata-rata 102,70 ms, pada metode berbasis *response time* didapatkan nilai *response time* dengan nilai rata-rata 67,56 ms, sedangkan pada metode berbasis *CPU Usage* pengujian *response time* nilai yang didapatkan adalah rata-rata 86,94 ms.

Tabel 4. 6 Hasil Pengujian *Response Time* Kategori *Rate High* (360)

No	HIGH		
	Round Robin (ms)	Response Time Based (ms)	CPU Usage Based (ms)
1	185,8	113,1	125,7
2	218,3	115,4	130,3
3	241,8	119,2	132,1
4	272,5	122,1	137,2
5	301,1	122,6	141,2
Rata-rata	243,90 (ms)	118,48 (ms)	133,28 (ms)

- Pada pengujian jumlah rate *high* pengujian *response time* dengan metode *round robin* adalah rata-rata 243,90 ms, pada metode berbasis *response time* didapatkan nilai *response time* rata-rata 118,48 ms, sedangkan pada metode berbasis *CPU Usage* pengujian *throughput* nilai yang didapatkan adalah rata-rata 133,28 ms.

4.1.4 Pengujian *CPU Usage*

Proses pengujian *CPU usage* dapat dilihat tabel 4.3, 4.4 dan 4.5. Satuan yang digunakan untuk *CPU Usage* adalah persentase (%). Data diperoleh dari hasil *record* dari masing masing *server* dan kemudian dilakukan perhitungan rata-rata dari nilai CPU *usage* yang didapat dari setiap *server*. Data diperoleh dengan menjalankan program *Agent Psutil* pada masing-masing *server*. Berikut hasil yang diperoleh:

Tabel 4. 7 Hasi Pengujian *CPU Usage* Kategori *Rate Low* (90)

	round robin (%)	CPU usage based (%)	response time based (%)
server 1	29,62	47,91	43,44
server 2	47,43	49,98	47,48
server 3	72,13	55,56	59,56

Berdasarkan tabel 4.7 dapat dijelaskan sebagai berikut :

- Pengujian dengan menggunakan metode round robin *server* 1 memiliki penggunaan *CPU usage* sebesar 29,62%, *server* 2 memiliki penggunaan *CPU usage* sebesar 47,43%, sedangkan pada *server* 3 memiliki penggunaan CPU usage sebesar 72,13%.
- Pengujian dengan menggunakan metode berbasis *CPU usage server* 1 memiliki penggunaan *CPU usage* sebesar 47,91%, *server* 2 memiliki penggunaan *CPU usage* sebesar 49,98%, sedangkan pada *server* 3 memiliki penggunaan CPU usage sebesar 55,56%.
- Pengujian dengan menggunakan metode berbasis *response time server* 1 memiliki penggunaan *CPU usage* sebesar 43,44%, *server* 2 memiliki penggunaan *CPU usage* sebesar 47,48%, sedangkan pada *server* 3 memiliki penggunaan CPU usage sebesar 55,56%.

Tabel 4. 8 Hasil Pengujian *CPU Usage Server* 2 Kategori *Rate Medium* (180)

	round robin	CPU usage based	response time based
server 1	43,46	60,95	53,46
server 2	60,61	65,22	57,88
server 3	88,02	70,33	72,67

Berdasarkan tabel 4.8 dapat dijelaskan sebagai berikut :

- Pengujian dengan menggunakan metode round robin *server* 1 memiliki penggunaan *CPU usage* sebesar 43,46%, *server* 2 memiliki penggunaan *CPU usage* sebesar 60,61%, sedangkan pada *server* 3 memiliki penggunaan CPU usage sebesar 88,02%.
- Pengujian dengan menggunakan metode berbasis *CPU usage server* 1 memiliki penggunaan *CPU usage* sebesar 60,95%, *server* 2 memiliki penggunaan *CPU usage* sebesar 65,22%, sedangkan pada *server* 3 memiliki penggunaan CPU usage sebesar 70,33%.
- Pengujian dengan menggunakan metode berbasis *response time server* 1 memiliki penggunaan *CPU usage* sebesar 53,46%, *server* 2 memiliki penggunaan *CPU usage* sebesar 57,88%, sedangkan pada *server* 3 memiliki penggunaan CPU usage sebesar 72,67%.

Tabel 4. 9 Hasil Pengujian *CPU Usage* Kategori *Rate High* (360)

	round robin (%)	CPU usage based (%)	response time based(%)
server 1	52,83	69,66	72,61
server 2	78,51	74,34	80,61
server 3	95,57	81,64	88,56

Berdasarkan tabel 4.9 dapat dijelaskan sebagai berikut :

- Pengujian dengan menggunakan metode round robin *server* 1 memiliki penggunaan *CPU usage* sebesar 52,83%, *server* 2 memiliki penggunaan *CPU usage* sebesar 78,51%, sedangkan pada *server* 3 memiliki penggunaan CPU usage sebesar 98,74%.
- Pengujian dengan menggunakan metode berbasis *CPU usage server* 1 memiliki penggunaan *CPU usage* sebesar 69,66%, *server* 2 memiliki penggunaan *CPU usage* sebesar 74,34%, sedangkan pada *server* 3 memiliki penggunaan CPU usage sebesar 81,64%.
- Pengujian dengan menggunakan metode berbasis *response time server* 1 memiliki penggunaan *CPU usage* sebesar 728,61%, *server* 2 memiliki penggunaan *CPU usage* sebesar 80,61%, sedangkan pada *server* 3 memiliki penggunaan CPU usage sebesar 88,56%.

4.1.5 Pengujian Distribusi *Traffic*

Proses pengujian pembagian *traffic* dapat dilihat dari tabel 4.6, 4.7 dan 4.8. Satuan yang digunakanuntuk pembagian *traffic* adalah jumlah koneksi . Data diperoleh dari hasil *record* dari masing masing *server*. Berikut hasil yang diperoleh:

Tabel 4. 10 Hasil Pengujian Pembagian *Traffic* Kategori *Low* (90)

	round robin (jumlah koneksi)	CPU usage based (jumlah koneksi)	response time based (jumlah koneksi)
server 1	300	471	405
server 2	300	348	328
server 3	300	111	167

Berdasarkan tabel 4.10 dapat dijelaskan sebagai berikut :

- Pengujian dengan menggunakan metode round robin, *server* 1 mendapat pembagian *traffic* sebanyak 300 koneksi, *server* 2 mendapat pembagian *traffic* sebanyak 300 koneksi, Pada *server* 3 mendapat pembagian *traffic* sebanyak 300 koneksi.
- Pengujian dengan menggunakan metode berbasis *CPU usage, server* 1 mendapat pembagian *traffic* sebanyak 471 koneksi, *server* 2 mendapat pembagian *traffic*

sebanyak 438 koneksi, Sedamgkan pada *server* 3 mendapat pembagian *traffic* sebanyak 111 koneksi.

- Pengujian dengan menggunakan metode berbasis *CPU usage, server* 1 mendapat pembagian *traffic* sebanyak 405 koneksi, *server* 2 mendapat pembagian *traffic* sebanyak 328 koneksi, sedangkan pada *server* 3 mendapat pembagian *traffic* sebanyak 167 koneksi.

Tabel 4. 11 Hasil Pengujian Pembagian *Traffic* Kategori *Medium* (180)

	round robin (jumlah koneksi)	CPU usage based (jumlah koneksi)	response time based (jumlah koneksi)
server 1	600	763	719
server 2	600	647	557
server 3	600	390	524

Berdasarkan tabel 4.11 dapat dijelaskan sebagai berikut :

- Pengujian dengan menggunakan metode round robin, *server* 1 mendapat pembagian *traffic* sebanyak 600 koneksi, *server* 2 mendapat pembagian *traffic* sebanyak 600 koneksi, Pada *server* 3 mendapat pembagian *traffic* sebanyak 600 koneksi.
- Pengujian dengan menggunakan metode berbasis *CPU usage, server* 1 mendapat pembagian *traffic* sebanyak 763 koneksi, *server* 2 mendapat pembagian *traffic* sebanyak 647 koneksi, Sedamgkan pada *server* 3 mendapat pembagian *traffic* sebanyak 390 koneksi.
- Pengujian dengan menggunakan metode berbasis *CPU usage, server* 1 mendapat pembagian *traffic* sebanyak 719 koneksi, *server* 2 mendapat pembagian *traffic* sebanyak 557 koneksi, sedangkan pada *server* 3 mendapat pembagian *traffic* sebanyak 524 koneksi.

Tabel 4. 12 Hasil Pengujian Pembagian *Traffic* Kategori *High* (360)

	round robin (jumlah koneksi)	CPU usage based (jumlah koneksi)	response time based (jumlah koneksi)
server 1	1200	1558	1313
server 2	1200	1273	1178
server 3	1200	769	870

Berdasarkan tabel 4.7 dapat dijelaskan sebagai berikut :

- Pengujian dengan menggunakan metode round robin, *server* 1 mendapat pembagian *traffic* sebanyak 1200 koneksi, *server* 2 mendapat pembagian *traffic* sebanyak 1200 koneksi, Pada *server* 3 mendapat pembagian *traffic* sebanyak 1200 koneksi.

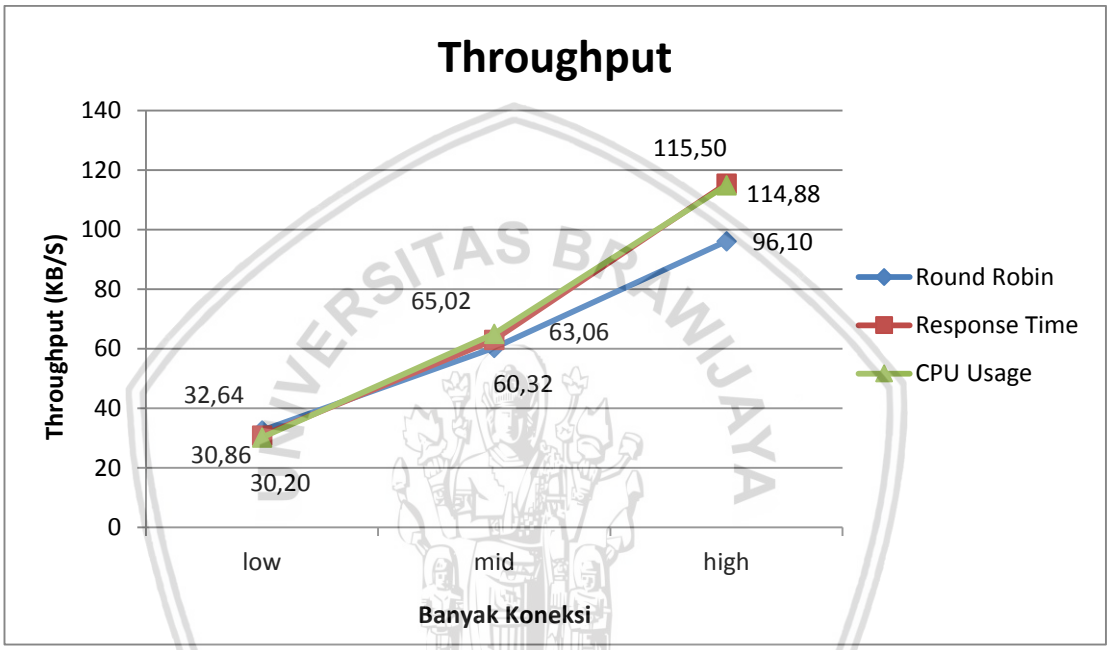
- Pengujian dengan menggunakan metode berbasis *CPU usage, server* 1 mendapat pembagian *traffic* sebanyak 1558 koneksi, *server* 2 mendapat pembagian *traffic* sebanyak 1273 koneksi, Sedamgkan pada *server* 3 mendapat pembagian *traffic* sebanyak 769 koneksi.
- Pengujian dengan menggunakan metode berbasis *CPU usage, server* 1 mendapat pembagian *traffic* sebanyak 1313 koneksi, *server* 2 mendapat pembagian *traffic* sebanyak 1178 koneksi, sedangkan pada *server* 3 mendapat pembagian *traffic* sebanyak 870 koneksi.

BAB 5 PEMBAHASAN

Pada bab ini membahas tentang perbandingan kinerja sistem *load balancing* dengan metode *round robin,* metode berbasis *CPU* dan Metode berbasis *response time* dari nilai hasil pengujian yang telah didapatkan.

5.1 Pembahasan Perbandingan Kinerja

5.1.1 Pengujian *Throughput*



Gambar 5. 1 Grafik Perbandingan Pengujian *Throughput*

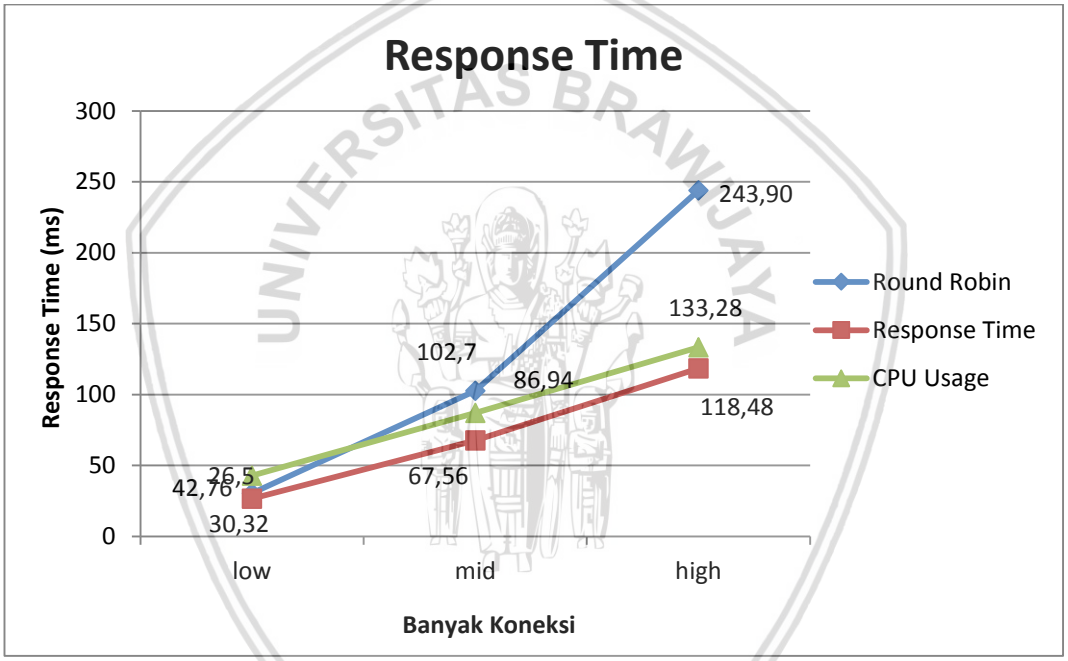
Grafik 5.1 merupakan grafik hasil pengujian *throughput* dari ketiga metode yang dirujuk dari tabel 4.1, 4.2 dan 4.3. *Throughput* adalah banyaknya informasi data yang dapat mengalir dari suatu perangkat ke perangkat lainnya dalam satuan waktu aktual tertentu. Tujuan *load balancing* sendiri adalah memaksimalkan *throughput.*

Pada ketiga skenario pengujian yang telah dilakukan bahwa metode *round robin* lebih baik hanya pada kategori rate *low* dengan banyak 90 *request.* Karena pada jumlah request yang rendah, kondisi *server* masih mampu melayani *request* dalam kondisi baik, yang membuat metode *round robin* mempunyai *throughput* yang lebih unggul. Karena memang metode *round robin* membagikan beban pada tiga *server* secara bergantian.

Sedangkan pada kategori *medium* dengan jumlah 180 *request* dan *high* dengan jumlah 360 *request*, nilai *throughput* metode berbasis *response time* lebih unggul, meskipun tidak ada perbedaan yang signifikan dengan hasil yang didapatkan dari metode berbasis *CPU usage.* karena memang metode berbasis *response time* merupakan representasi dari metode berbasis *CPU usage,* tetapi

dilihat dari parameter yang berbeda. Jika pada metode berbasis *CPU usage* dilihat dari kondisi *CPU server*, tetapi pada metode berbasis *response time* dilihat dari parameter *response time* dari *server*. Karena kondisi *CPU* dari *server* juga akan mempengaruhi *response* time dari *server* tersebut, dimana kedua metode ini melakukan pendistribusian *traffic* dengan melihat kondisi dari *server* yang membuat *throughput* yang dihasilkan unggul dibandingkan metode *round robin*. Metode *round robin* pada jumlah *request* yang tinggi memiliki nilai *throughput* yang rendah dibandingkan dengan metode berbasis *response time* dan metode berbasis *CPU usage*. karena metode *round robin* mendistribusikan *traffic* tanpa melihat kondisi dari *server* yang membuat kondisi *server* akan mengalami kelebihan beban yang mengakibatkan *throughput* yang dihasilkan rendah.

5.1.2 Pengujian *Response Time*



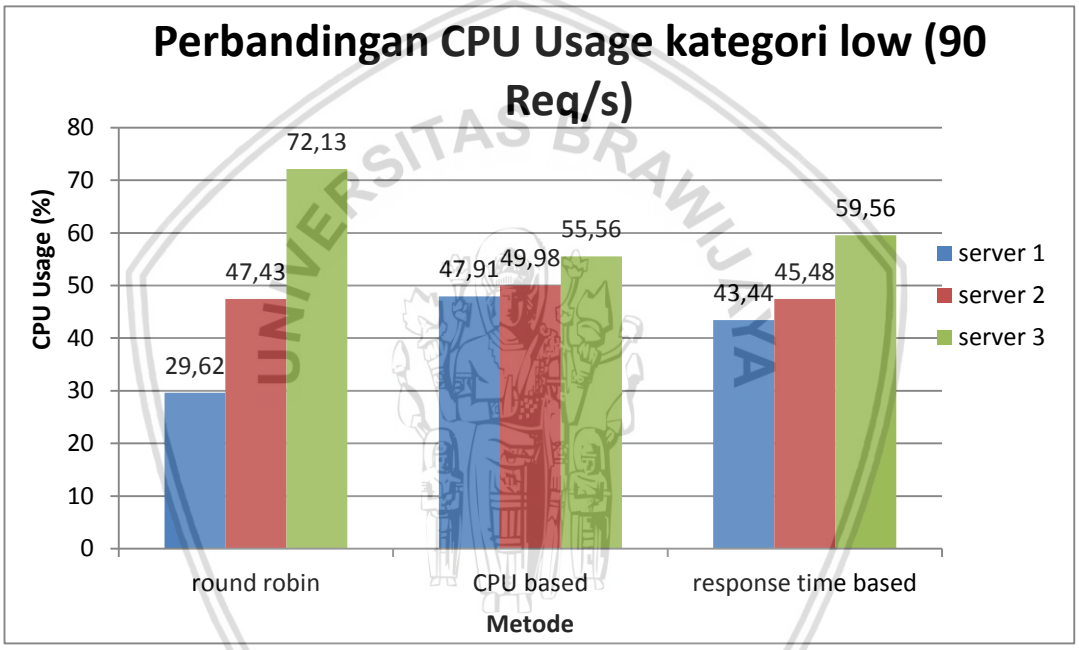
Gambar 5. 2 Grafik Perbandingan Pengujian *Response Time*

Grafik 5.2 merupakan grafik hasil pengujian *response time* dari metode *round robin,* metode berbasis *CPU* dan Metode berbasis *response time* yang dirujuk dari tabel 4.4, 4.5 dan 4.6. *Response time* adalah waktu tanggap yang diberikan ketika *client* melakukan *request* atau mengirim permintaaan ke *server*. Tujuan *load balancing* adalah mengurangi *response time,* dengan dua atau lebih *server* yang saling berbagi beban *traffic*, masing-masing akan berjalan lebih cepat karena beban tidak berada pada 1 *server* saja.

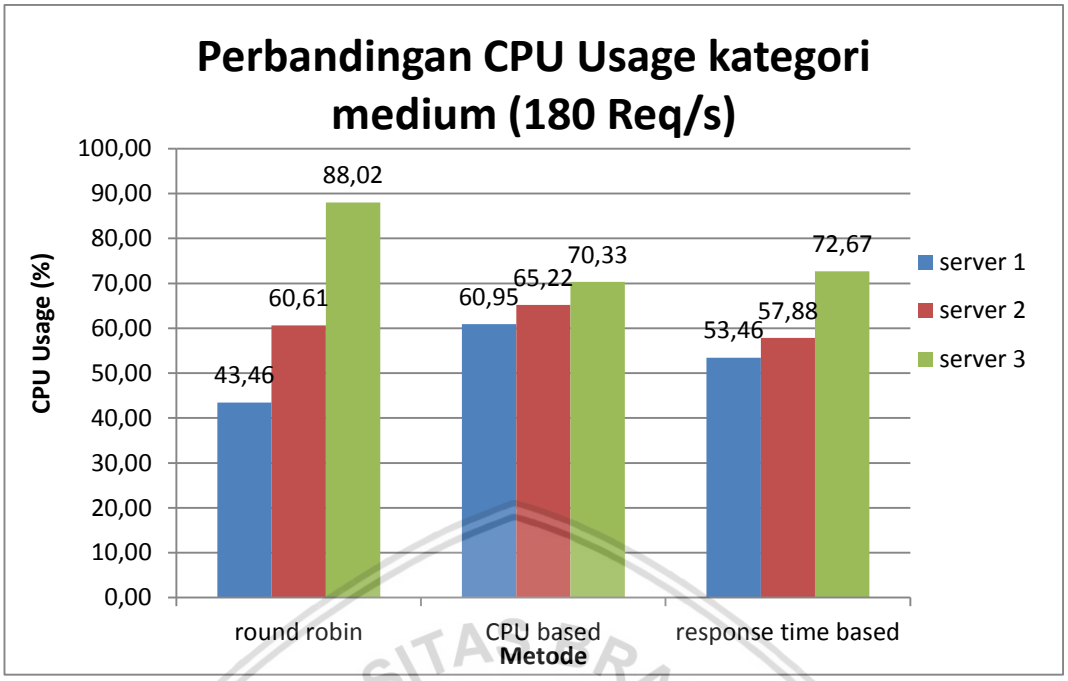
Nilai *response time* bertambah seiring dengan perubahan kategori *rate,* semakin tinggi kategori *rate,* maka semakin tinggi nilai *response time* yang didapatkan. Pada penelitian ini metode berbasis *response time* memiliki *response time* tercepat dari semua kategori pengujian. Karena memang beban *traffic* akan diarahkan ke *server* yang memiliki waktu respon tercepat. Dapat dilihat dari gambar 5.2, kenaikan nilai *response time* dari metode berbasis *response time* dan

metode berbasis *CPU usage* pada setiap kategori *rate,* tidak setinggi dari metode *round robin.* Karena memang pembagian beban pada kedua metode ini akan dilakukan sesuai dengan kondisi atau kapasitas dari *server* yang membuat kinerja *server* menjadi seimbang dan mencegah *server* dalam kondisi *overload* dan dapat meyebabkan waktu respon dari *server* menjadi lambat. Berbeda dengan metode *round robin* yang membuat *server* dengan kapasitas *core* rendah mengalami kelebihan beban dan membuat waktu respon dari *server* menjadi lambat. Dapat dilihat dari gambar 5.2, bahwa metode *round robin* mengalami kenaikan *response time* yang tinggi dari pengujian kategori *medium (180 request)* ke pengujian kategori *high (360 request) .*

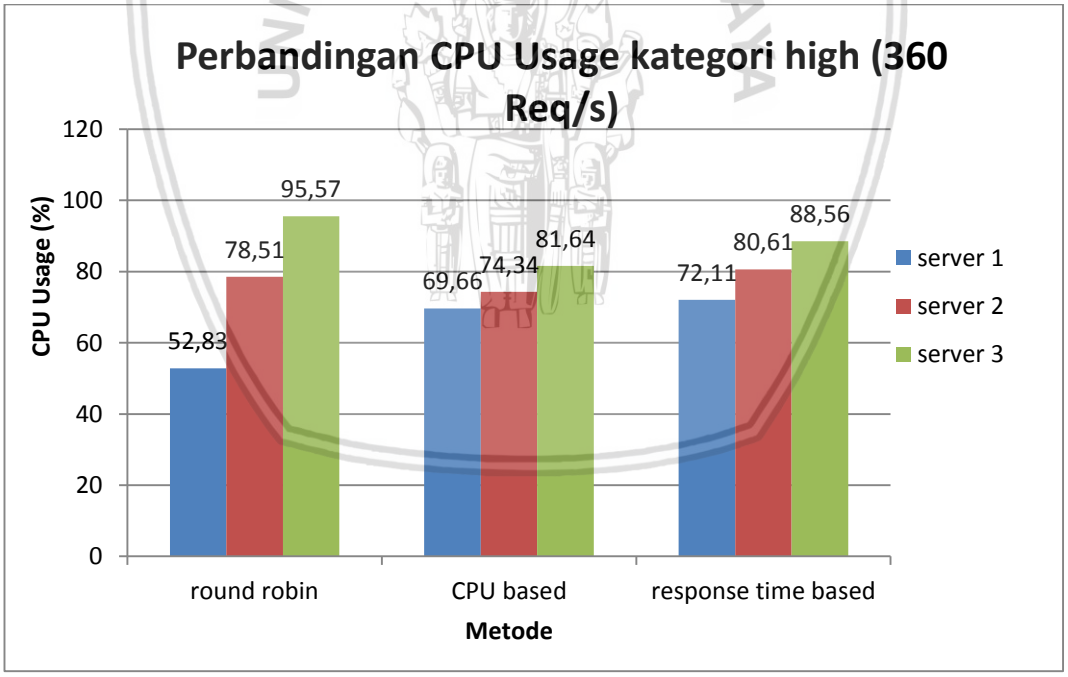
5.1.3 Pengujian *CPU Usage*



Gambar 5. 3 Grafik Pengujian *CPU Usage* pengiriman 90 Req/s



Gambar 5. 4 Grafik Pengujian *CPU Usage* pengiriman 180 Req/s



Gambar 5. 5 Grafik Pengujian *CPU Usage* pengiriman 360 Req/s

Grafik 5,3, 5,4 dan 5,5 merupakan grafik hasil pengujian *CPU usage* setiap *server*. *CPU usage* adalah tingkat beban *CPU* pada *server* ketika melayani *client*. *Load balancing* berfungsi untuk menyeimbangkan beban *server*. Sehingga *server* tidak akan terlalu terbebani. Semakin kecil *CPU usage,* maka semakin baik kinerja *server* dalam melayani *client* pada suatu jaringan.

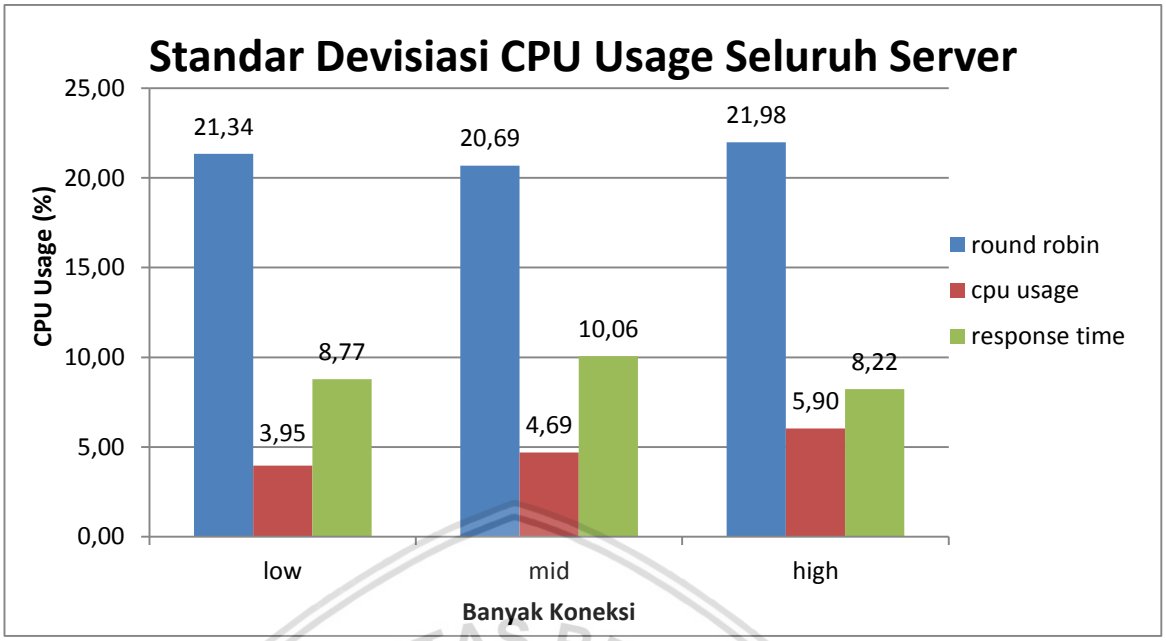
Bisa dilihat dari ketiga grafik tersebut bahwa metode *round robin* memiliki penggunaan *CPU* dengan rentan lumayan jauh dari ketiga *server*, karena metode *round robin* akan mendistribusikan *traffic* secara merata tanpa melihat kondisi dari *server*. Berbeda dengan metode berbasis *CPU usage* dan metode berbasis *response time* yang melihat kondisi dari *server* untuk pendistribusian *traffic*. Bisa dilihat dari semua kategori *rate,* untuk metode berbasis *CPU usage* dan metode berbasis *response time* selisih dari ketiga *server* tidak terlalu jauh seperti yang ada pada metode *round robin.* Penggunaan *CPU* pada *server* 1 juga memiliki penggunaan yang lebih tinggi dari metode *round robin* dan *server* 3 memiliki penggunaan *CPU* lebih rendah dari metode *round robin*.

Untuk mengetahui penggunaan *CPU yang* lebih seimbang dan stabil dari metode-metode tersebut, dibutuhkan perhitungan standar deviasi. Standar deviasi adalah ukuran yang digunakan untuk mengukur jumlah variasi atau sebaran sejumlah nilai data.

Tabel 5. 1 Standar Deviasi *CPU Usage* Seluruh *Server*

Round Robin				
Rate	Server 1	Server 2	Server 3	STDDEV
90	29,62	47,43	72,13	21,34
180	47,46	60,61	88,02	20,69
360	52,83	78,51	95,57	23,00
CPU Usage Based				
Rate	Server 1	Server 2	Server 3	STDDEV
90	47,91	49,98	55,56	3,95
180	60,95	65,22	70,33	4,69
360	69,66	74,34	81,64	5,90
Response Time Based				
Rate	Server 1	Server 2	Server 3	STDDEV
90	39,44	43,48	67,56	8,77
180	50,46	67,88	75,67	10,06
360	72,11	80,61	88,56	8,22

Tabel 5.1 merupakan nilai yang diambil dari pengujian *CPU usage* setiap *server*. Kemudian diakukan perhitungan standar deviasi dan *average CPU usage* seluruh *server* sesuai dengan kategori *rate*. Barikut adalah grafik standar deviasi.

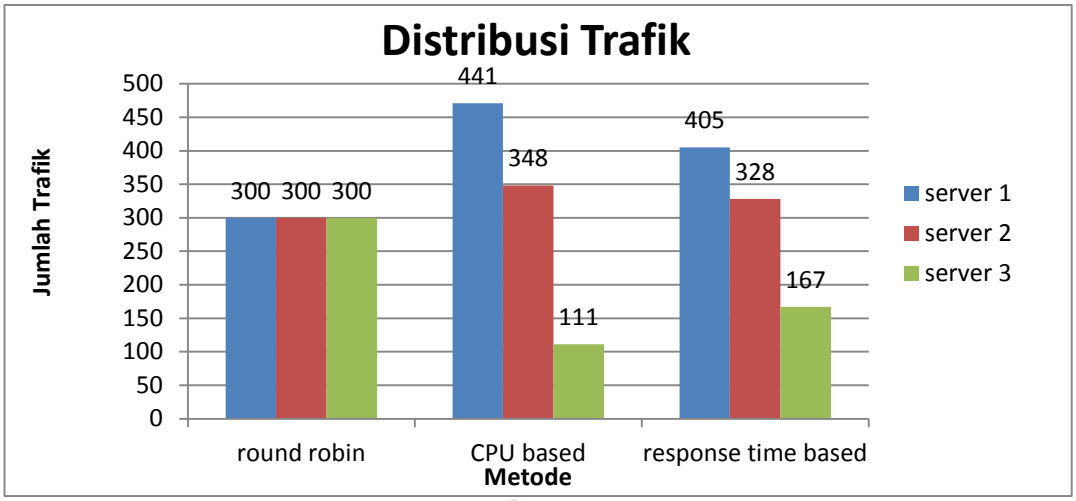


Gambar 5. 6 Grafik Perbandingan Standar Deviasi *CPU Usage* Seluruh *Server*

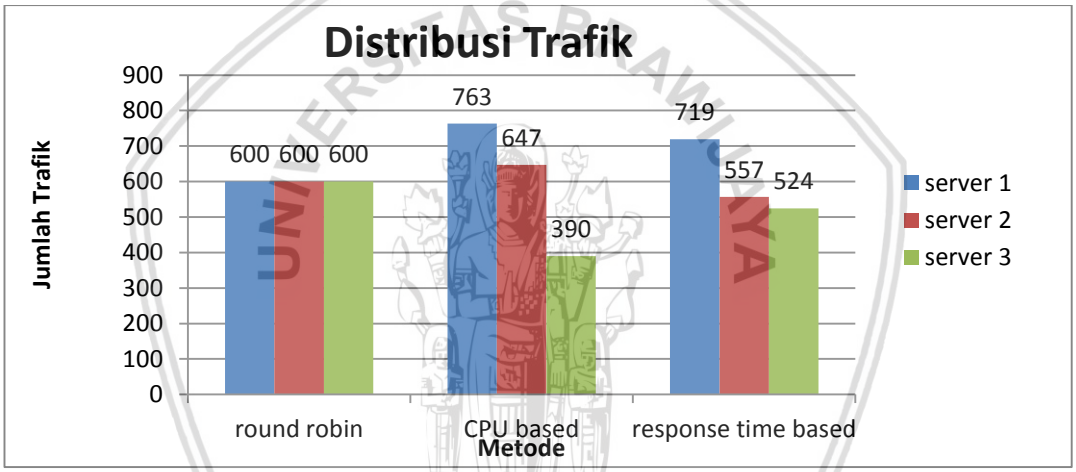
Gambar 5.6 merupakan standar deviasi dari seluruh *server*. Metode berbasis *CPU usage* memiliki nilai standar deviasi yang paling rendah. Hal ini menandakan bahwa metode berbasis *CPU usage* mendistribusikan *traffic* ketiga *server* secara adil dengan melihat kapasitas *CPU* dari *server.* Jika dibandingkan dengan metode *round robin* yang memiliki nilai tinggi, karena memang *round robin* tidak memperdulikan kapasitas dari *server* yang membuat pembagian beban tidak seimbang. Hal ini yang menyebabkan kondisi *CPU* yang memiliki jumlah *core* kecil akan lebih terbebani dibandingkan dengan *server* yang memiliki jumlah *core* lebih besar. Sedangkan pada metode berbasis *response time* memiliki standar deviasi sedikit lebih tinggi dari metode berbasis *CPU usage*. Metode berbasis *response time* melakukan *load balancing* dengan mengarahkan *traffic* pada *server* yang memiliki respon tercepat. Kondisi *CPU* dari *server* akan mempengaruhi waktu respon dari *server*. Oleh karena itu standar deviasi dari metode berbasis *response time* tidak jauh beda dengan apa yang didapatkan oleh metode berbasis *CPU usage*.

5.1.4 Pengujian Distribusi *Traffic*

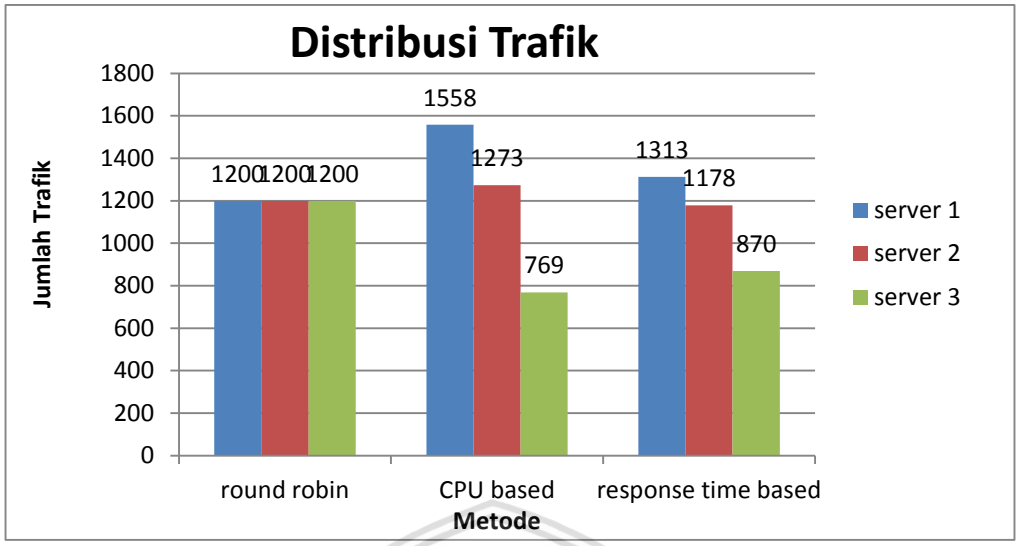
Setiap metode memiliki pendistribusian *traffic* yang berbeda-beda. Pengujian distribusi *traffic* ditujukan sebagi bukti bahwa metode *round robin*, metode berbasis *CPU* dan metode berbasis *response time* dapat berjalan dengan baik dan juga sebagi bukti dari hasil standar deviasi yang dijelaskan pada gambar 5.7. Data di rujuk dari tabel 4.10, 4.11 dan 4.12. Hasil yang didapatkan adalah sebagai berikut.



Gambar 5. 7 Grafik Distribusi *Traffic* Pengiriman 90 Req/s



Gambar 5. 8 Grafik Distribusi *Traffic* Pengiriman 180 Req/s



Gambar 5. 9 Grafik Distribusi *Traffic* Pengiriman 360 Req/s

Gambar 5.7, 5.8 dan 5.9 menjelaskan bahwa dari semua kategori *rate* metode *round robin* akan mendistribusikan *traffic* secara bergantian, meskipun kondisi *server* tidak sama yang menyebabkan kondisi ketiga *server* memiliki beban yang tidak seimbang. Sedangkan pada metode berbasis *CPU usage* dan metode berbasis *response time* akan lebih banyak mendistribusikan *traffic* pada *server* yang memiliki jumlah *core* lebih tinggi dibandingkan pada *server* yang memiliki jumlah *core* rendah.

BAB 6 PENUTUP

Berdasarkan metodologi penelitian yang sudah disusun sebelumnya, bab ini merupakan tahap akhir dalam melakukan penelitian setelah melakukan pengujian dan analisis hasil pengujian dari sistem yang sudah dirancang.

6.1 Kesimpulan

1. Pada implementasi metode *load balancing*, *SDN controller* dijadikan sebagai *load balancer* yang menerapkan metode *round robin*, metode berbasis *CPU* dan metode berbasis *response time*. *Controller* yang digunakan adalah *POX controller*. Khusus untuk metode berbasis *CPU* terdapat sebuah agen yang disebut *Agen Psutils* yang berjalan pada setiap *server.* Agen tersebut mengirimkan informasi penggunaan *CPU* pada setiap *server* sehingga *SDN controller* dapat mengetahui penggunaan *CPU* pada setiap *server*. Dari implementasi tersebut, teknologi *load balancing* dapat berjalan dengan baik dan melakukan pembagian *traffic* jaringan dengan benar, berdasarkan komputasi dari metode *round robin*, metode berbasis *CPU* dan metode berbasis *response time.*
2. Kinerja metode *round robin* pada pengujian *throughput* lebih unggul dibandingkan metode berbasis *CPU usage* dan metode berbais *response time* pada pengujian kategori *rate low* dengan nilai rata-rata 32,64 KB/s. Sedangkan pada pengujian kategori *rate* medium dan *high* metode berbasis *response time* lebih unggul dibandingkan metode *round robin* dan metode berbais *CPU usage* dengan nilai rata-rata 65,02 KB/s pada *rate medium* dan 115,50 KB/s pada *rate high*.
3. Kinerja metode berbasis *response time* pada pengujian *response time* lebih unggul dibandingkan dengan metode berbasis *CPU usage* dan metode *round robin* dari semua kategori *rate* dengan nilai rata-rata 30,32 ms pada *rate low*, 67,56 ms pada *rate medium* dan 118,48 ms pada *rate high.* Karena memang beban *traffic* akan diarahkan ke *server* yang memiliki waktu respon tercepat.
4. Kinerja metode berbasis CPU usage pada pengujian *CPU usage server* memiliki pembagian beban lebih stabil dibandingkan dengan metode r*ound robin* dan metode berbasis *response time.* Bisa dilihat dari standart deviasi, bahwa metode berbasis *CPU usage* memiliki nilai rata-rata paling rendah yaitu 3,95% pada *rate low*, 4,69% pada *rate medium* dan 5,90% pada *rate high*.

6.2 Saran

Ada beberapa saran untuk para peneliti yang ingin melakukan pengembangan pada penelitian ini antara lain :

1. Melakukan perbandingan metode *load balancing* yang lainnya untuk mengetahui kinerja yang dihasilkan.
2. Melakukan implementasi dengan menggunakan *controller* selain pox.
3. Melakukan pengujian dengan parameter pengujian yang lebih banyak lagi.
4. Diharapkan dalam penelitian selanjutnya menggunakan *server* nyata atau *real.*

DAFTAR PUSTAKA

Arianto, E., & Sholeh, M. (2014). Implementation of Load Balancing Two Line ISP Using Router OS. *Jurnal JARKOM, 2*.

Ellrod, C. (2010). *Load Balancing – Least Connection*. Retrieved Oktober 18, 2016, from https://www.citrix.com/blogs/2010/09/02/load-balancing-least-connections/

Julianto, R. (2017). Implementasi Load Balancing di Web Server Menggunakan Metode Berbasis Sumber daya CPU Pada Software Defined Networking. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*.

Kadir, A. (2003). *Pengenalan Sistem Informasi.* Andi Yogyakarta.

Kartadie, R., & Utami, E. (2014). Prototipe Infrasturkture Software Defined Network Dengan Protokol Openflow Menggunakan Ubuntu Sebagai Kontroler. *JURNAL DASI, 15*(1).

Lara, A., & Kolasani, A. (2014). Network Innovation using OpenFlow: A Survey. *CSE Journal Articles*.

Mahdiyanah, N. (2016). Implementasi Load balancing di Web Server dengan Algoritma Round robin pada Software Defined Network. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*.

Mosberger, D. (1998). httperf - A Tool for Measuring Web Server Kinerjance.

Open Networking Foundation. (2009). OpenFlow Switch Specification. Retrieved Oketober 19, 2016, from https://www.opennetworking.org/sdn-resources/sdn-definition

Psutil Documentation. (2018, Maret 5). Retrieved from psutil documentation: http://psutil.readthedocs.io/en/latest/

Sirajuddin, & Affandi, A. (2012). Rancang Bangun Server Learning Management System. *JURNAL TEKNIK ITS*.

Ummah, I., & Abdillah, D. (2016). Perancangan Simulasi Jaringan Virtual Berbasis Software-Define Networking. *Ind. Journal on Computing, I*(1).

Zhong, H., & Fang, Y. (2016). An Efficient SDN Load Balancing Scheme Based on Server Response Time. *Future Generation Computer Systems*.

SDN Control Plane & SDN Data Plane. (2018, Januari 4). Diambil kembali dari sdx central: https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/

Moniruzzaman, M. (2015). A High Availability Cluster Model Combined with Load Balancing and Shared Storage Technologi for Web Servers. *International Journal of Grid Distribution Computing.*