repository.ub.ac.id

MEKANISME PEMBOBOTAN *SERVER* MENGGUNAKAN ALGORITME *FUZZY* PADA SISTEM *LOAD BALANCING* DI *SOFTWARE DEFINED NETWORK*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Hasbi Razzak
NIM: 135150201111288

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

PENGESAHAN

MEKANISME PEMBOBOTAN SERVER MENGGUNAKAN ALGORITME *FUZZY* PADA SISTEM *LOAD BALANCING* DI *SOFTWARE DEFINED NETWORK*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh :
Hasbi Razzak
NIM: 135150201111288

Skripsi ini telah diuji dan dinyatakan lulus pada
23 Juli 2019
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Widhi Yahya, S.Kom., M.Sc.	M. Ali Fauzi, S.Kom., M.Kom.
NIK: 2016078911211001	NIK: 201502 890101 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

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

UNIVERSITAS BRAWIJAYA

PERNYATAAN ORSINILITAS

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, 23 Juli 2019

Hasbi Razzak

NIM: 135150201111288

PRAKATA

Alhamdulillah, segala puji syukur kita ucapkan atas kehadirat Allah SWT yang dengan atas rahmat dan karunia-Nya penulis mampu meyelesaikan Skripsi dengan judul "MEKANISME PEMBOBOTAN SERVER MENGGUNAKAN ALGORITME FUZZY PADA SISTEM LOAD BALANCING DI SOFTWARE DEFINED NETWORK" dengan baik. Skripsi ini diajukan untuk memenuhi syarat serta mendapatkan gelar sarjana di Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.

Tugas akhir ini dapat terselesaikan karena atas bantuan, dukungan, dorongan serta bimbingan dari berbagai pihak. Untuk itu penulis sangat berterima kasih kepada:

1. Bapak Widhi Yahya S. Kom, M. Sc sebagai dosen pembimbing 1 yang telah memberikan ilmu dan saran serta banyak meluangkan waktunya untuk penulis terkait pengerjaan pada penelitian ini.
2. Bapak M. Ali Fauzi, S. Kom, M. Kom sebagai dosen pembimbing 2 yang telah membantu penulis serta memberikan bimbingan dan arahan kepada penulis terkait penelitian ini.
3. Orang tua penulis yang selalu mendo'akan dan menyemangati penulis dalam proses pengerjaan skripsi ini. Oleh karenanya penulis selalu sadar untuk menyelesaikan penelitian ini.
4. Muharrom Abdillah, S.Kom. dan Muhammad Sholeh S.Kom. yang selalu memberikan motivasi dan saran terkait penelitian dan penulisan skripsi.
5. Teman-teman angkatan 2013 Program Studi Teknik Informatika/ Ilmu Komputer yang tidak dapat penulis sebutkan satu persatu disini atas seluruh saran, bantuan dan motivasi yang telah diberikan.

Terakhir, penulis menyadari bahwa skripsi ini masih banyak kekurangan dan jauh dari sempurna. Oleh karena itu, saran dan kritik terkait skripsi ini yang sifatnya membangun untuk memperbaiki mutu penulisan sangat diharapkan penulis untuk penelitian selanjutnya. Penulis berharap dengan penelitian ini dapat bermanfaat.

Malang, 23 Juli 2019

Penulis

hasbirazzak@gmail.com

repository.ub.ac.id

ABSTRAK

Hasbi Razzak, Mekanisme Pembobotan Server Menggunakan Algoritme *Fuzzy* Pada Sistem *Load Balancing* Di *Software Defined Network*

Pembimbing: Widhi Yahya, S. Kom, M. Sc dan M. Ali Fauzi, S. Kom., M. Kom

Server merupakan tempat penyimpanan berbagai macam layanan. Kualitas layanan pada Server dipengaruhi oleh kinerja dari server tersebut. Ketika sebuah web server hanya memiliki satu server saja, maka server itu akan mengalami overload jika server menerima traffic yang tinggi. Untuk mengatasi masalah tersebut dapat menggunakan multiple server. Oleh karena itu, dibutuhkan sebuah metode didalam multiple server sebagai pembagian beban kinerja agar setiap server dapat berjalan dengan stabil. Metode yang dapat digunakan adalah teknologi *load balancing*. Teknologi *load balancing* dapat diimplementasikan pada *software defined network* agar mencapai kondisi sistem yang optimal dan beban kerja didistribusikan secara merata di antara server serta menurunkan waktu eksekusi program.

Pada penelitian ini sistem load balancing diimplementasikan menggunakan algoritme *Fuzzy* yang diterapkan pada Arsitektur Software Defined Network. Algoritme *Fuzzy* pada sistem load balancing bertujuan untuk meneruskan *request* terhadap server berdasarkan bobot server terendah. Kemudian akan dibandingkan dengan algoritme *Least Connection* yang meneruskan *request* terhadap server berdasarkan koneksi server terkecil dan juga algoritme *Response Time* yang meneruskan *request* terhadap server berdasarkan *response time* server terkecil. Penelitian ini akan melakukan pengujian dengan 3 skenario yaitu rate 90 untuk *low*, rate 180 untuk *medium* dan rate 360 untuk *high*. Kemudian dilakukan perbandingan dengan algoritme Response time (RT) dan algoritme Least Connection (LC) dengan parameter distribusi trafik, CPU dan Memory Usage, Response Time dan Troughput.

Pada pengujian dengan beban request dengan jumlah rate 360 algoritme Fuzzy lebih membebankan pendistribusian trafik pada server yang bobot server terendah. Kemudian pada pengujian *memory usage*, algoritme RT memiliki penggunaan memory terendah di setiap server yaitu server 1 18,7%, server2 32,4% dan server3 62,2%. Selanjutnya pada pengujian dengan beban pada *rate* 360 algoritme RT memiliki response time yang terkecil yaitu 393 m/s pada pengujian *troughput* algoritme RT memiliki troughput terbesar yaitu 93,6 KB.

Kata kunci: Load Balancing, Software Defined Network, Fuzzy, Agen Psutil, Pox *Controller*

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

ABSTRACT

Hasbi Razzak, Mekanisme Pembobotan Server Menggunakan Algoritme *Fuzzy* Pada Sistem *Load Balancing* Di *Software Defined Network*

Pembimbing: Widhi Yahya, S. Kom, M. Sc dan M. Ali Fauzi, S. Kom., M. Kom

One of the uses of the server is to store various kinds of services. The service quality on the server is affected by the performance of the server. When a web server only has one server, the server will become overload if the server receives high traffic. To overcome this problem, we need a method that can distribute the performance load so that each server can run stable. The method used in this research is load balancing. Load balancing can be implemented on Software Defined Networks to achieve optimal system conditions. The workloads are distributed equally between servers to reduce program execution time.

In this study load balancing systems are implemented using Fuzzy algorithms that are applied to the Software Defined Network Architecture. This algorithm will be compared with the Least Connection and Response Time algorithms. Fuzzy algorithm on load balancing systems aims to forward requests to the server based on the lowest server weight. The Least connection will forward the request to the server based on the smallest server connection, while the Response Time algorithm is based on the smallest response time from the server. This study will conduct testing with 3 scenarios: 90 rates for the low category, 180 rates for the medium category and 360 rates for the high category. Then a comparison with the algorithm that has been mentioned is based on parameters; traffic distribution, CPU and Memory Usage, Response Time and Throughput.

In testing with load requests with a number of 360 rates, Fuzzy algorithms charge more for the distribution of traffic on servers with high resources. Then in testing the memory usage, the RT algorithm has the lowest memory usage on each server which is 18.7% on server 1, 32.4% on server 2 and 62.2% on server 3. Then on testing with load at rate 360, the RT algorithm has the smallest response time is 393 m/s. While throughput testing, RT algorithms have the largest throughput of 93.6 KB.

Keywords*: Load Balancing, Software Defined Network, Fuzzy, Agen Psutil, Pox controller*

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Server merupakan media sebagai tempat penyimpanan berbagai macam layanan. Adapun layanan seperti Web yang menggunakan protokol HTTP yang dapat diakses melalui jaringan komputer. Kualitas layanan *Server* dipengaruhi oleh kinerja server tersebut. Sebuah server dapat mengalami overload yang disebabkan oleh ribuan hingga jutaan request dalam waktu bersamaan melebihi kemampuan *server* (Rosalia, 2016). Untuk menghindari terjadinya overload, maka diperlukan *multiple server*. *Multiple server* dapat menerapkan sebuah metode yang berfungsi sebagai pembagi beban kinerja agar setiap *server* dapat berjalan dengan stabil. Metode yang dapat digunakan untuk *multiple server* adalah *load balancing* sebagai pendistribusi beban server.

Load balancing merupakan teknologi yang berfungsi mendistribusikan atau membagi beban kinerja pada sekumpulan server ketika menangani *request* dari pengguna berupa layanan atau perangkat pada jaringan (Rijayana, 2005). Sistem *load balancing* mampu mendistribusikan beban kinerja pada setiap *server*, untuk menghindari terjadinya *overload*. Untuk mencapai sistem *load balancing* yang optimal, dibutuhkan sebuah teknik yang dapat menghemat waktu eksekusi program dan mendsitribusikan beban kinerja server secara merata. Pada umumnya metode yang digunakan pada sistem *load balancing* merupakan algoritme yang tidak berubah keadaannya seperti *Randomized* dan *Round robin* (LI & Chang, 2007).

Pada penelitian-penelitian terkait sistem *load balancing* terdapat sebuah arsitektur baru yaitu *Software Defined Network* atau SDN. SDN adalah jaringan yang bersifat dinamis, memungkinkan terpusat, *programmable*, serta terdapat mekanisme yang mampu memisahankan *control plane* dengan *data plane* secara fisik, sehingga *administrator* dapat mengelola dan mengontrol sumber daya virtual secara langsung tanpa harus mengenal teknologi perangkat jaringan (Shin, Nam, & Kim, 2012). Oleh sebab itu, untuk dapat menangani sistem pada jaringan yang kompleks, SDN menjadi pilihan yang tepat (Hu, Hao, & Bao, 2014). Sehingga banyak penelitian terkait SDN di antaranya dengan judul "Weighted Round-Robin Load Balancing Using Software Defined Networking" (Sabiya & Singh, 2016). Penelitian tersebut mendistribusikan beban kinerja server berdasarkan bobot kinerja server untuk menentukan server yang melayani *request.* Dari hasil penelitian tersebut, pendistribusian beban akan diarahkan pada server yang telah ditentukan berdasarkan bobot kinerja server dengan nilai 60 % ke server 1, 30% ke server 2 dan 10% ke server 3. Penelitian tersebut server 1 akan mendapatkan beban kinerja yang lebih besar dibandingkan server lainnya. Oleh sebab itu metode *Weighted Round-robin* kurang efisien serta tidak melihat kondisi sebenarnya dari masing-masing server dan dapat menyebabkan server *overload.* Kemudian kinerja server mengacu pada banyak penggunaan sumber daya misalnya, CPU, *memory, network* dan *disk* (Citrix, 2016). Oleh karena itu,

dibutuhkan sebuah mekanisme yang lebih efisien serta dapat mendistribusikan beban yang mengacu pada banyak parameter untuk menentukan bobot dari server. Dari kebutuhan tersebut, penelitian ini akan menggunakan algoritme *Fuzzy* sebagai pengambil keputusan terhadap server yang tepat untuk menangani *request.* Algoritme *Fuzzy* mampu melakukan kategorisasi berdasarkan kondisi server dengan parameter yang ditentukan yaitu CPU, Memory dan Disk.

Oleh karena itu, pada penelitian ini akan melakukan penelitian untuk pendistribusian beban server pada sistem *load balancing* yang diterapkan pada arsitektur SDN yang berjudul "Mekanisme Pembobotan *Server* Menggunakan Algoritme *Fuzzy* Pada Sistem *Load balancing* Di Arsitektur *Software Defined Network*". Dengan adanya penelitian ini, diharapkan mampu mengatasi kekurangan dari algoritme pada penelitian sebelumnya dan dapat memberikan kinerja yang optimal mengacu pada kondisi sumber daya server. Kemudian hasil dari pengujian penelitian ini akan dibandingan dengan beberapa algoritme yaitu *Least Connection* yang membagi koneksi berdasarkan koneksi terkecil dan *Response Time* yang membagi koneksi berdasarkan *response time* terkecil yang dibandingkan dengan algoritme fuzzy berdasarkan sumber daya server terendah. Perbandingan tersebut bertujuan untuk mengetahui kinerja server agar terhindar dari *overload* dengan parameter pengujian distribusi trafik, penggunaan sumber daya, kecepatan waktu tanggap (*response time*) dan *throughput*.

1.2 Rumusan Masalah

Berdasarkan pendauluan, masalah yang dperoleh adalah sebagai berikut ini:

1. Bagaimana implementasi algoritme *Fuzzy* pada sistem load balancing untuk mekanisme pembobotan Server yang diterapkan di arsitektur *Software Defined Network*?
2. Bagaimana kinerja *load balancing server* dengan menggunakan algoritme *Fuzzy* di arsitektur *Software Defined Network* yang dibandingkan dengan algoritme *Response time* dan algoritme *Least Connection*?

1.3 Tujuan

Berdasarkan rumusan masalah diatas, maka tujuan dari penelitian ini yaitu:

1. Mengimplementasikan sistem *Load balancing Server* menggunakan *algoritme Fuzzy* untu mekanisme pembobotan server yang diterapkan di arsitektur *Software Defined Network*.
2. Menguji kinerja sistem *Load balancing* yang menggunakan *algoritme Fuzzy* untuk pembobotan server yang diterapkan di arsitektur *Software Defined Network*.
3. Membandingkan kinerja *load balacing* server yang diterapkan pada *Software Defined Network* menggunakan *algoritme Fuzzy* terhadap algoritme *Response Time* dan algoritme *Least Connection*.

1.4 Manfaat

Berikut ini merupakan manfaat dari penelitian ini yang diharapkan adalah:

1. Dapat mengimplementasikan *algoritme Fuzzy* pada *load balancing server* untuk mekanisme pembobotan server yang diterapkan di arsitektur *Software Defined Network*.
2. Dapat mengetahui kinerja *load balancing* server menggunakan *algoritme Fuzzy* yang dibandingkan dengan algoritme *Response time* dan algoritme *Least connection*.
3. Dapat berkontribusi dalam mengembangkan *algoritme* yang dapat diterapkan pada sistem *load balancing* di arsitektur SDN.

1.5 Batasan Masalah

Untuk dapat mencapai tujuan yang diharapkan penelitian ini, maka ditentukanlah batasan dari masalah tersebut yaitu:

1. Penelitian ini menggunakan *web server virtual machine (*VM) sebanyak 3 buah*,* sebuah SDN *controller* dan sebuah *SDN switch* yang mengunakan laptop*.*
2. Data yang dihasilkan pada pengujian penelitian ini merupakan data dari lingkungan kerja penelitian.
3. Sistem *Load balancing* ini hanya dapat melayani *Hyper Text Transfer Protocol* (HTTP).

1.6 Sistematika Pembahasan

Adapun sistematika pembahasan dari penelitian ini terbagi dalam beberapa bab meliputi:

BAB I : Pendahuluan

Pada bagian ini peneliti akan membahas latar belakang yang menjadi dasar dari penelitian ini. Diawali dengan permasalahan yang ada disertai solusi untuk permasalahan tersebut. Kemudian peneliti membuat rumusan masalah tentang pertanyaan terkait penelitian. Selanjutnya tujuan dan manfaat dari hasil penelitian ini, serta ditentukannya batasan masalah dari penelitian ini agar tercapai penelitian yang diharapkan peneliti.

BAB II : Landasan Kepustakaan

Pada bagian ini peneliti akan membahas hal-hal yang mendasari sistem *load balancing* menggunakan algoritme *Fuzzy* yang diterapkan di SDN. Adapun langkah-langkah pada bagian ini meliputi: kajian pustaka, sistem *Load balancing*, logika *Fuzzy,* SDN*, Web Server, Psutil* dan Httperf.

BAB III : Metodologi

Pada bagian ini peneliti, akan membahas alur penelitian yang akan dilakukan yaitu : Studi literatur membahasxdasar teori danmreferensi pada penelitianmini, rekaya kebutuhan membahas kebutuhan yang diperlukan penelitian ini, perancangan membahas tentang bagaimana merancanga sistem yang akan dibangun, implementasi membahas tentang bagaimana mengimplementasikan sistem yang sudah dirancang, pengujian membahas tentang hasil yang didapatkan dari sistem yang dibangun dan penarikan kesimpulan dari penelitian serta saran terkait penelitian.

BAB IV : Rekayasa Kebutuhan

Pada bagian ini, peneliti akan membahas kebutuhan sistem yang akan dibangun adalah gambaran umum sistem dan deskripsi kebutuhan sistem yang akan dibutuhkan pada perancangan dan implemtasi sistem.

BAB V : Perancangan

Pada bagian ini, peneliti akan membahas perancangan meliputi : perancangan arsitektur SDN, Perancangan *load balancing*, perancangan algoritme *Fuzzy* dan perancangan algoritme pada agen *Psutil*.

BAB VI : Implementasi

Pada bagian ini, peneliti akan membahas implementasi meliputi : konfigurasi perangkat, implementasi arsitektur SDN, implementasi *load balancing,* implementasi algoritme *Fuzzy* dan implementasi algoritme pada agen *Psutil*.

BAB VII : Pengujian

Pada bagian ini, peneliti akan membahas pengujian terhadap kinerja sistem *load balancing* menggunakan algoritme fuzzy, serta dibandingkan dengan algoritme *least connection* dan *response time*. Pengujian dilakukan menggunakan empat parameter yaitu: distribusi traffik, CPU dan *memory usage*, *response time* dan *troughput,* serta terbagi kedalam tiga pengujian yaitu: pengujian fungsionalitas, pengujian tanpa beban dan pengujian dengan beban.

BAB VIII : PENUTUP

Pada bagian ini, peneliti akan menarik kesimpulan tentang penelitian ini dan memuat saran untuk pengembangan studi di masa depan.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Dalam tahap ini, kita akan menjelaskan metode dan objek yang digunakan pada penelitian sebelumnya. Kemudian, dibandingkan dengan penelitian ini sebagai acuan penelitian.

Pada penelitian sebelumnya yang berjudul "Implementasi Load Balancer Berdasarkan Server Status pada Arsitektur Software Defined Network (SDN)" (Islahul Ardy, 2018). *Controller* pada penelitian tesebut mampu mendistribusikan beban kinerja server mengacu pada keadaan atau status server. Keadaan server adalah bobot dari kinerja yang diatur oleh *Controller* sesuai dengan kemampuan masing-masing server yang digunakan. *Controller* tersebut menentukan nilai bobot (weight), serta *Controller* dapat menentukan beban dari tiap server untuk menangani *request*. Oleh karena itu, merujuk pada penelitian tersebut untuk memenuhi kebutuhan sistem penelitian yang akan dilakukan, maka akan menggunakan *Controller* SDN yang sama.

Kemudian pada penelitian yang telah dilakukan dengan judul "A *Fuzzy* Synthetic Evaluation Algorithm with Dynamic Weight for SDN" (Wang & Guo, 2017). Penelitian tersebut membahas sistem *load balancing* di arsitektur SDN untuk mekanisme pembobotan jalur koneksi yang mengacu pada bobot dari koneksi untuk menentukan jalur terpendek. Tujuan dari penelitian tersebut, meningkatkan kinerja sistem *load balancing* untuk mendistribusikan beban trafik secara dinamis berdasarkan jalur terpendek. Untuk itu agar kebutuhan sistem penelitian ini terpenuhi, maka akan menggunakan algoritme yang sama untuk mekanisme pembobotan server pada pendistribusian beban trafik berdasarkan penggunaan sumber daya server seperti CPU, *memory* dan disk.

Selanjutnya pada penelitian sebelumnya dengan berjudul "Implementasi Load balancing Menggunakan Metode Berbasis Sumber Daya CPU pada Software Defined Networking" (Julianto, 2017). Penelitian tersebut menggunakan sebuah agen yang ditanam di server. Agen berfungsi untuk mengirim hasil data penggunaan CPU yang digunakan oleh setiap server. Kemudian mendistribusikan trafik berdasarkan penggunaan CPU paling rendah yang dimiliki tiap server. Fungsi dari agen tersebut, digunakan kebutuhan penelitian ini terpenuhi. Oleh sebab itu, agen digunakan pada penelitian ini untuk mengirim informasi sumber daya server ke *Controller*.

Pada penelitian sebelumnya dengan judul "Analisis Perbandingan Kinerja Metode Load Balancing Berbasis CPU Dengan Berbasis Response Time Pada Software Defined Network" (Abdillah, 2018). Pada penelitian tersebut menguji kinerja algoritme yang diterapkan pada load balancing server, pengujian tersebut menggunakan parameter CPU *usage* dan *troughput.* Hasil dari pengujian penggunaan CPU metode tersebut, mempunyai nilai rendah dalam pengunaan CPU dibandingkan metode perbandingannya. Kemudian pada pengujian *troughput*, algoritme *response time* lebih baik dibandingkan algoritme

perbandingannya. Oleh sebab itu, parameter pengujian yang digunakan penelitian tersebut, juga digunakan pada pengujian ini.

Berikut merupakan penjelasan mengenai rencana dari penelitian ini dan akan dibandingkan dengan penelitian sebelumnya seperti pada Tabel 2.1.

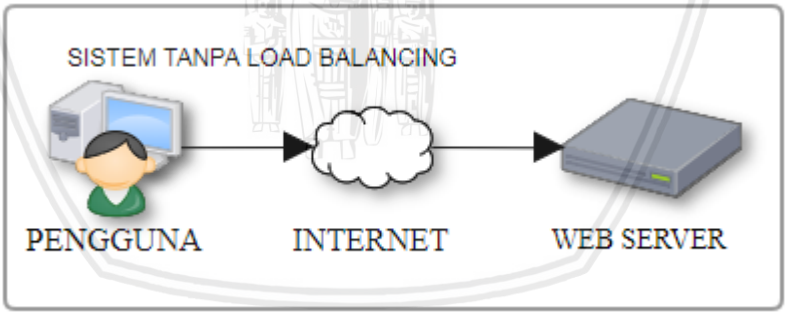
Tabel 2. 1 Kajian Pustaka

No	Nama Peneliti, Tahun, dan Judul	Persamaan	Perbedaan	
			Penelitiann sebelumnya	Rencana penelitian
1	Lalu Fani Islahul Ardy, 2018. Implementasi Load Balancer Berdasarkan Server Status pada Arsitektur Software Defined Network (SDN)	Menerapkan *load balancing* di arsitektur SDN	Mengimplemen tasikan *load balancing* berdasarkan status dari server.	Menerapkan algoritme *Fuzzy* pada *load balancing* berdasarkan parameter CPU *usage, memory usage dan disk*.
2	Wang & Guo, 2017. A *Fuzzy* Synthetic Evaluation Algorithm with Dynamic Weight for SDN.	Menerapkan arsitektur SDN pada sistem *load balancing*.	Menerapkan algoritme *Fuzzy* pada *load balancing* sebagai pendistribusian trafik berdasarkan jalur terpendek.	Menerapkan algoritme *Fuzzy* pada *load balancing* sebagai pendistribusian trafik berdasarkan penggunaan sumber daya server terendah.
3	Riski julianto, *2017.* Implementasi Load balancing Menggunakan Metode Berbasis Sumber Daya CPU pada SDN.	Menerepkan controller POX di arsitektur SDN pada untuk pendistribusian beban server.	Menggunakan *Agen Psutils* untuk mendapatkan informasi CPU pada server di dalam cluster.	Menggunakan *Agen Psutils* untuk mendapatkan informasi penggunaan sumberdaya server didalam cluster.

4	Muharrom Abdillah, 2018. Analisis Perbandingan Kinerja Metode Load Balancing Berbasis CPU Dengan Berbasis Response Time Pada Software Defined Network.	Menggunakan parameter pengujian penggunaan CPU dan *response time*.	Perbandingan kinerja sistem *load balancing* yang diterapkan di arsitektur SDN pada algoritme berbasis CPU dengan metode *response time*.	Menerapkan *load balancing* pada arsistektur SDN menggunakan algoritme *Fuzzy* yang dibandingkan pada metode *Least connection* metode dan *Response time.*

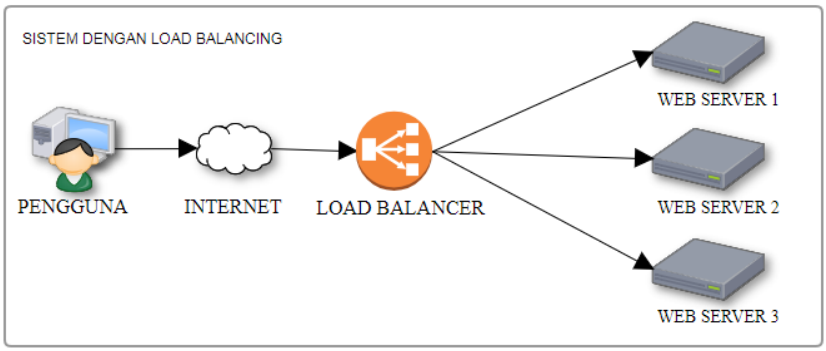
2.2 Load Balancing

Sistem pendistribusian beban server atau disebut *load balancing* adalah teknologi yang mampu menyeimbangkan pekerjaan pada dua atau banyak komputer, perangkat jaringan, sumber daya lain, yang berfungsi mengoptimalkan sumber daya, respons time dan *troughput* (Karimi, 2009). *Load balancing* merupakan sebuah metode yang dapat membagi beban pekerjaan ke banyak server untuk mengoptimalkan penggunaan sumber daya seperti CPU, Memory, Disk dan lainya. Metode *load balancing* dapat menghindari kesalahan atau *overload* pada salah satu sumber daya sehingga meningkatkan *high availability* pada sistem.



Gambar 2. 1 Sistem Tanpa Load Balancer

Sistem *load balancing* dapat diimplementasikan pada layanan seperti HTTP yang menggunakan server sebagai tempat untuk mengakomodasi permintaan klien yang disebut *Web server*. Dalam layanan HTTP tradisional atau tanpa menggunakan *load balancing* yang ditunjukkan Gambar 2.1, jika ada klien yang mengakses layanan, maka hanya dilayani oleh *Web server* tunggal. Sehingga jika terjadi kesalahan pada Web Server tersebut maka akan berdampak pada sistem secara keseluruhan. Namun, jika sebuah sistem menggunakan *load balancing* seperti Gambar 2.2, maka request yang dilakukan oleh klien akan dilayani oleh banyak *Web server* yang tersedia. Sehingga, apabila ada Web Server tidak dapat dijalankan maka Web Server lainnya yang akan menangani *request*.



Gambar 2. 2 Sistem dengan Load Balancer

Berdasarkan Gambar 2.2 adalah layanan *web* yang menerapkan *load balancing*. Lalu ada penyeimbang beban bersama dengan tiga *Server Web* yang akan menanggapi permintaan pelanggan. Memuat penyeimbang sebagai pengatur lalu lintas untuk menentukan *Server Web* yang dapat melayani lalu lintas yang datang tergantung pada kemampuan *algoritme* yang diterapkan.

2.3 Logika Fuzzy

Pada algoritme *Fuzzy* memiliki mekanisme yang disebut logika *Fuzzy* yang dapat diterapkan dalam sistem. Bukan hanya sistem sederhana, sistem seperti jaringan komputer, sistem kontrol, sistem terintegrasi, dan *workstation* berbasis akuisisi data. Logika *Fuzzy* dapat digunakan untuk menyelesaikan masalah tidak linier atau biner. Adapun alasan penelitian ini menerapkan logika *Fuzzy* karena: dapat dimengerti, mempunyai toleransi terhadap data presisi, serta dapat memodelkan fungsi yang kompleks dan juga mampu menggunakan teknik-teknik kendali yang saling bekerja sama berdasarkan bahasa alami secara konvensional (Rahmawati, 2015).

2.3.1 Himpunan *Fuzzy*

Berdasarkan logika Fuzzy, terdapat sebuah himpunan *Fuzzy* yang dapat mengelompokan variabel linguistik dan diekspresikan oleh fungsi keanggotaan. logika *Fuzzy* memiliki dua macam himpunan, yaitu (Saelan, 2009). Pertama, himpunan *crips* yang menyakatakan kondisi dari himpunan *Fuzzy*. Himpunan Fuzzy dapat disebut himpunan tegas yang memliki simbol yaitu (μ) dengan nilai (1) menyakan "ya" dan (0) "tidak. Selanjutnya himpunan samar yang dapat memiliki nilai keanggotaan berbeda dalam satu himpunan.

Himpunan *Fuzzy* umumnya memiliki dua atribut, yang pertama Linguistik merupakan penamaan yang mewakili suatu keadaan dengan bahasa yang dapat dimengerti seperti: tinggi, pendek dan sedang. Kemudian yang kedua adalah numerik yang merupakan dinyatakan dengan nilai seperti: 20, 50, 70, 100 dan lainnya.

Selanjutnya agar dapat memahami mekanisme *Fuzzy*, hal-hal yang perlu diketahui yaitu variabel *Fuzzy* dan semestra pembicara. Adapun variabel *Fuzzy*

tediri dari beberapa himpunan *Fuzzy* seperti variabel umur dengan tiga himpunan *Fuzzy* yaitu: muda, parobaya dan tua. Sedangkan, semesta pembicara merupakan nilai variabel fuzzy yang dijinkan untuk dioperasikan.

2.3.2 Fungsi Keanggotaan

Fungsi keanggotaan adalah himpunan *Fuzzy* yang dapat ditentukan dengan fungsi linier, fungsi segitiga (*triangel*), trapesium (*trapezoidal*) atau fungsi Gauss (*Gaussian*) (Wulandari, 2011).

2.3.3 Fungsi Implikasi

Terdapat dua macam Fungsi implikasi pada metode *Fuzzy* pertama adalah Min yang berfungsi untuk memotong output dari himpunan *Fuzzy*, kedua adalah Dot yang berfungsi untuk menskala output dari himpunan *Fuzzy*.

2.3.4 Sistem Aturan *Fuzzy*

Pada mekanisme logika fuzzy, diperlukan adanya pendekatan logika *Fuzzy* yang diimplementasikan ke dalam 3 proses yaitu:

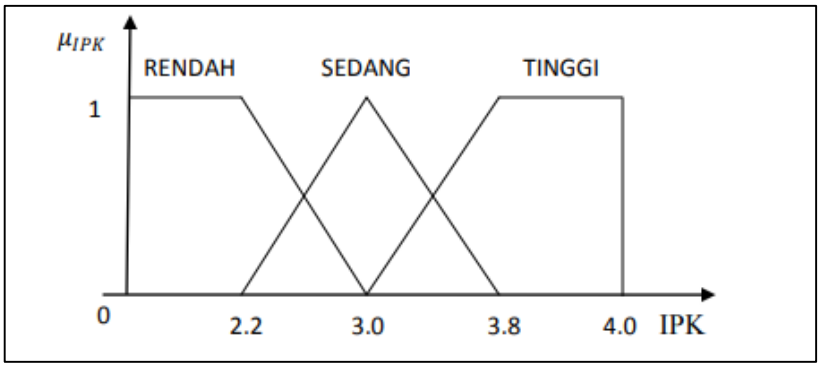
1. Fuzzifikasi
2. *Evaluasi Rule* (Inferensi)
3. Defuzzifikasi

2.3.5 *Fuzzy* Inference System Mamdani (FIS Mamdani)

Metode FIS Mamdani memiliki kelebihihan jika dibandingkan metode penalaran lainnya yang bersifat intuitif, dapat mencakup berbagai bidang dan juga dapat disesuaikan dengan informasi manusia sebagai variable input. FIS Mamdani adalah algoritme pengambil keputusan yang menggunakan teori dari himpunan *Fuzzy* untuk memetakan variable input kedalam variabel output. Seorang bernama Ebrahim Mamdani telah menemukan metode tersebut pada tahun 1975. Agar dapat output dari metode *Fuzzy* diperlukan 4 proses, sebagai berikut (Imam, Wijana, & Prawiti, 2010):

2.3.5.1 Pembentukan Himpunan *Fuzzy*

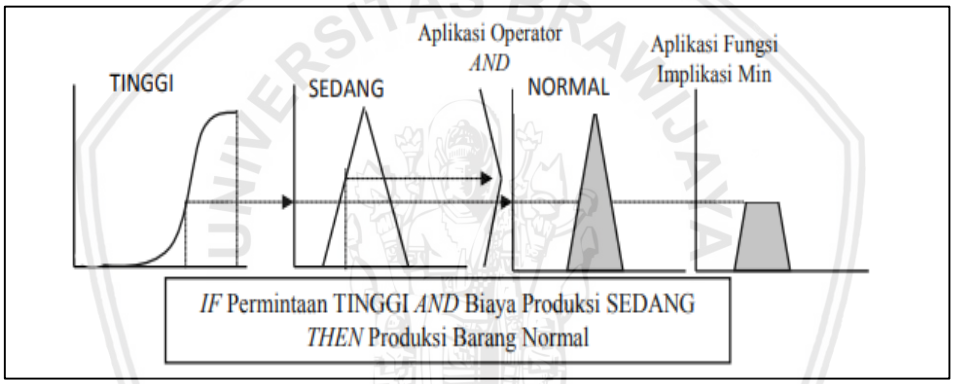
Pada tahap ini, dibuat beberapa himpunan yang menyatakan kondisi dari setiap variable yang telah dibuat. himpunan tersebut, bertujuan untuk mencari nilai dari variable output. Berdasarkan Gambar 2.3 menjelaskan himpunan Fuzzy dari variable IPK. Variable tersebut memiliki tiga himpunan seperti: sedang, tinggi dan rendah.



Gambar 2. 3 Contoh Himpunan IPK

2.3.5.2 Aplikasi fungsi implikasi

Setelah melalui proses pembentukan himpunan, langkah selanjutnya merupakan aturan yang digunakan pada aplikasi fungsi implikasi di metode mamdani adalah Min dalam Gambar 2.4 berikut.



Gambar 2. 4 Contoh Metode Min

2.3.5.3 Komposisi aturan

Setelah aturan dibuat untuk digunakan pada tahap aplikasi fungsi implikasi, proses selanjutnya merupakan komposisi aturan dengan mengunakan metode Max untuk proses *Fuzzy inference system* Mamdani.

2.3.5.4 *Penegasan* (Defuzzifikasi)

Setelah melalui beberapa proses, proses defuzzifikasi bertujuan untuk mendapatkan nilai yang didapatkan melalui proses komposisi aturan berdasarkan aturan *Fuzzy*. Ada lima metode yang bisa digunakan untuk proses defuzzifikasi, namun metode yang memiliki nilai *mean square error* (MSE) yang paling rendah merupakan metode terbaik (Shaleh, 2015). Adapun metode yang digunakan pada FIS mamdani adalah Centroid.

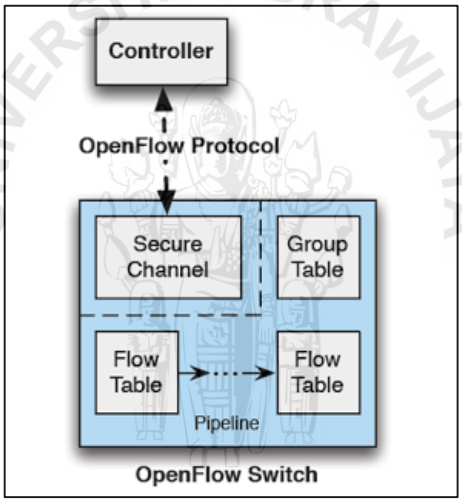
2.4 *Sofware Defined Network* (SDN)

Pada sistem *load balancing* terdapat teknologi baru yang dapat mendisain, manajemen dan implementasi jaringan yang rumit dan kompleks adalah *Sofware Defined Network* (SDN). mekanisme yang dimiliki oleh SDN dapat

memisahankan *control plane* dengan *data plane* secara fisik, sehingga *network administrator* dapat mengelola dan mengontrol sumber daya virtual secara *real time* (Kartadie & Utami, 2014).oleh sebab itu , setiap perangkat dapat diatur hanya melalui perangkat *Controller* saja. Pada penerapannya dibutuhkan API sebagai penghubung perangkat jaringan pada sebuah *controller* disesuaikan dengan kebutuhan yang diperlukan. Munculnya SDN berawal dari kebutuhan sebuah *software* yang dapat mengatur perangkat jaringan.

2.4.1 *OpenFlow*

Seiring berkembangnya paradigma dari Software Defined Network (SDN), pengembang maupun praktisi dalam jaringan berusaha dan berlomba-lomba untuk mengembangkan protokol komunikasi antara Controller dengan Switch pada arsitektur SDN tersebut. OpenFlow merupakan salah satu protokol yang mampu menjembatani pengiriman data antara Controller dengan Switch. Tidak hanya itu, OpenFlow juga memiliki subsistem yang tersusun atas tiga bagian yaitu OpenFlow Protocol, Flow Table dan Secure Channel.



Gambar 2. 5 Open*flow Switch*

OpenFlow mendukung protokol komunikasi yang aman melalui *Secure Channel* yang terlihat dalam Gambar 2.3. Sehingga dengan teknologi tersebut memungkinkan data yang dikirimkan protokol tersebut terhindar dari aktifitas yang berdampak buruk bagi jaringan. Kemudian, OpenFlow juga dapat membuat pengelompokan dari Flow Table yang sudah terbentuk sehingga memudahkan pengelolaan Flow pada OpenFlow Switch (Open Networking Foundation, 2009).

2.4.2 SDN *Controller*

SDN *Controller*nmerupakan komponen utama yang mampu mengatur trafik data pada jaringan. Dengan menggunakan protokol komunikasi seperti OpenFlow, *Controller*mdapat mengtrol dan mengelolaaaliran datanmenggunakan Switch. Sudah banyak aplikasi yangdditerapkan ke dalamc*Controller* seperti *routing system*, *high availability control*, *data center, firewall system* dan lain-lain. Untuk menentukan SDN *Controller* yang akan digunakan tentu harus dilihat

dari kebutuhan dari penelitian. Saat ini juga banyak sekali SDN Controller salah satunya adalah POX SDN Controller. Controller ini ditulis dengan menggunakan bahasa pemrograman yang populer yaitu Python (McCauley, M, 2015).

2.5 Web Server

Web Server adalah *host* atau komputer yang memiliki layanan Web yang dijalankan dengan protokol HTTP dan HTTPS. Web Server menggunakan model *client*/server. Secara sederhana ketika penguna mengakses alamat dari *Web Server* seperti "http://www.google.com" maka HTTP *request* tersebut akan di *response* oleh Web Server. Oleh karena itu, Web Server dapat digunakan untuk mengakses layanan yang diinginkan pengguna seperti *Web page* dengan menggunakan protokol komunikasi yang tersedia.

2.6 Psutil

Psutil adalah salah satu library yang disediakan oleh Python. Fungsi dari library ini yaitu memantau aktifitas dan proses yang berjalan pada sumber daya seperti CPU, *memory*, disk, *network* dan lain-lain. Selain dari itu, library tersebut dapat mewakili dari penggunaan *tool* umum pada UNIX seperti ps, top, netstat, ifconfig dan lain-lain. (Psutil Documentation, 2018). Dalam penelitian ini, library ini akan digunakan untuk mendapatkan data server berupa informasi penggunaan sumber daya server.

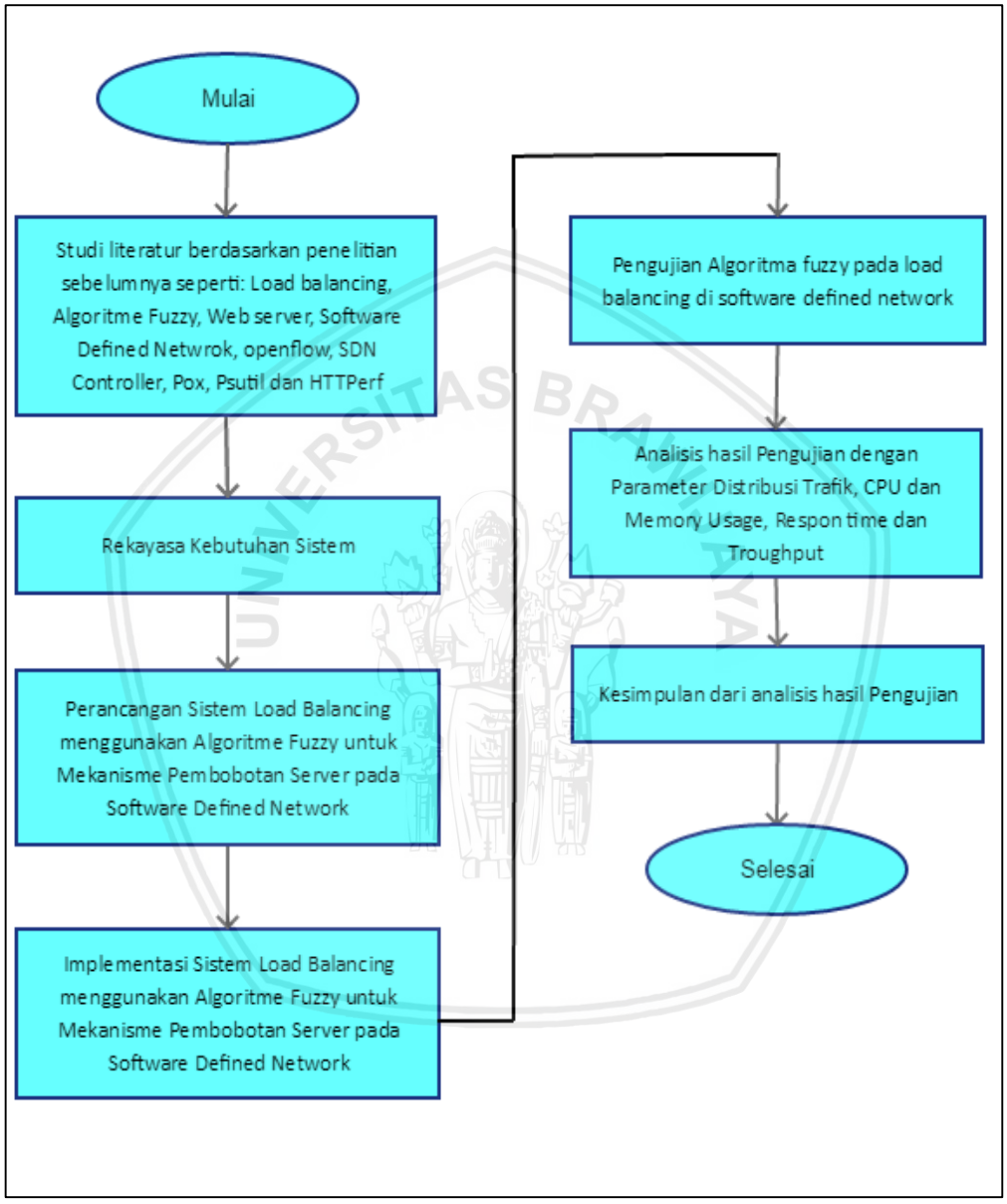
2.7 HTTPerf

Ketika membangun sebuah sistem, kita akan dihadapkan pada pertanyaan apakah sistem tersebut sudah layak untuk digunakan? Atau sejauh mana batas kemampuan dari sistem tersebut? Dan lain sebagainya. Pada penelitian ini, sistem load balancing harus dapat mendistribusikan trafik jaringan secara optimal. Oleh karena itu, kita dapat menggunakan HTTPerf dalam tahap pengujian dari sistem load balancing tersebut.

HTTPerf merupakan sebuah alat yang dapat menghasilkan *workload* pada layanan HTTP. Pada prosesnya, HTTPerf dapat me-*request* ke dengan beberapa jumlah koneksi yang telah dikonfigurasi, sehinggakpengguna HTTPerf dapat mengetahui kinerja dari sistem *load balancing* tersebut (Mosbergen & Jin, 1998).

BAB 3 METODOLOGI

Bab metodologi membahas langkah-langkah yang akan dijelaskan pada proses penelitian ini. Berikut adalah langkah-langkah metodologis dari penelitian yang akan dilakukan.



Gambar 3. 1 Diagram Alir Metode Penelitian

Berdasarkan Gambar 3.1 peneliti akan menguraikan metode yang digunakan oleh peneliti dalam penelitian ini. Pada awalnya, peneliti mencari dasar teoritis. Kemudian dilanjutkan pada tahap rekayasa kebutuhan sistem, seperti deskripsi sistem umum, persyaratan fungsional dan non-fungsional. Kemudian, kebutuhan sistem, tahap perancangan sistem dan implementasi sistem. Selain itu, melakukan fase pengujian serta pembahasan dari hasil yang diperoleh serta memuat kesimpulan dari hasil pengujian.

3.1 Studi Literatur

Dalam tahap ini, peneliti akan menjelaskan hal-hal mendasar terkait sistem yang akan dibangun. Adapun langkah-langkah pada bagian ini meliputi: kajian pustaka, sistem *Load balancing*, logika *Fuzzy*, SDN, *Web Server*, Psutil dan Httperf.

3.2 Rekayasa Kebutuhan

Dalam tahap ini, peneliti akan menguraikan kebutuhan pada gambaran umum sistem serta deskripsi kebutuhan sistem yang dibutuhkan pada tahap perancangan dan tahap implemtasi sistem.

3.3 Perancangan

Dalam tahap ini, peneliti akan membahas perancangan meliputi : perancangan arsitektur SDN, perancangan *load balancing*, perancangan algoritme *Fuzzy* dan perancangan algoritme pada agen *Psutil*.

3.4 Implementasi

Dalam tahap ini, peneliti akan membahas tahapan implementasi yang diawalai dengan konfigurasi perangkat, implementasi arsitektur SDN, implementasi *load balancing*, implementasi algoritme *Fuzzy* dan implementasi algoritme pada agen *Psutil*.

3.5 Pengujian

Dalam tahap ini, peneliti akan membahas pengujian dari kinerja sistem *load balancing* menggunakan algoritme *Fuzzy*, serta dibandingkan terhadap dua algoritme yaitu *Least connection* (LC) dan *Response time* (RT). Adapun parameter pengujian yaitu: distribusi trafik, *troughput,* CPU *usage, memory usage dan response time*, serta terbagi kedalam tiga pengujian yaitu: pengujian tanpa beban, pengujian dengan beban dan fungsionalitas.

3.6 Kesimpulan

Dalam tahap ini, peneliti melakukan penarikan kesimpulan setelah semua tahapan dilakukan pada penelitian ini terpenuhi. Kemudian peneliti akan memuat kesimpulan pada bab penutup terkait penelitian yang sudah dilakukan dan saran untuk studi selanjutnya.

BAB 4 REKAYASA KEBUTUHAN

4.1 Gambaran Umum Sistem

Sistem ini akan dibangun menggunakan metode *load balancing* untuk melakukan pendistribusian beban pada server dengan menggunakan algoritme *Fuzzy*. Algoritme fuzzy akan melakukan perhitungan atau perbandingan dari beberapa server dengan menggunakan beberapa variabel. Masing-masing server akan mengirimkan informasi pada *controller* menggunakan Psutil. Psutil akan dijalankan pada masing- masing server, kemudian *controller* nilai bobot server yang dikirim oleh server untuk selanjutnya dilakukan perhitungan dan perbandingan.

Pada penelitian yang dilakukan, metode *load balancing* akan di terapkan pada arsitektur *Software Defined Network* yang terbagi menjadi 4 komponen. Komponen-komponen tersebut akan sama-sama terhubung dan membentuk sebuah sistem *load balancing*. Adapun komponen-komponen tersebut yaitu web server, *switch,controller* dan *Psutil*, yang akan dijelaskan sebagai berikut :

- *Web server* yaitu perangkat pada sistem *load balancing* yang berfungsi sebagai penyedia layanan web. Pada komponen ini, *web server* melayani permintaaan secara bergantian terhadap layanan yang diterima. Masing-masing server memiliki layanan web server(tersinkronisasi).
- Switch yaitu komponen yang bertugas untuk meneruskan paket data yang dikirim. Switch akan meneruskan paket yang dikirim sesuai dengan perintah yang diberikan oleh *controller*.
- *Controller* yaitu suatu perangkat yang bisa dibilang sebagai “otak” dari suatu sistem. *controller* pada arsitektur SDN ini berperan sangat penting dengan menjalankan algoritme *Fuzzy* untuk perhitungan dan perbandingan kondisi dari suatu server yang kemudian menentukan server yang menangani permintaan dari klien.
- Agen *Psutil* yaitu program yang bertugas untuk mengirim data berupa informasi sumber dari daya suatu server yang kemudian data tersebut akan dikirimkan pada *controller*. Program ini akan dipasangkan dan dijalankan pada masing-masing server yang nantinya akan dikirimkan ke *controller* untuk dilakukan perhitungan dan perbandingan server.

4.2 Deskripsi Kebutuhan sistem

Dalam tahap ini menjelaskan terkait kebutuhan-kebutuhan yang digunakan guna memberi kemudahan dalam tahap perancangan dan pengimplementasian sistem yang dibangun. Adapun kebutuhan yang digunakan untuk membangun sistem ini, seperti: Kebutuhan fungsional dan non-fungsional, serta kebutuhan lingkungan kerja sistem.

4.2.1 Kebutuhan Fungsional

Pada Kebutuhan fungsional ini menjelaskan apa saja yang dapat dilakukan oleh sistem. Setiap sistem memiliki komponen-komponen yang harus dapat berjalan dengan baik sesuai dengan tugasnya masing-masing. Berikut merupakan kebutuan fungsional dari sistem ini:

4.2.1.1 SDN *Controller*

1. *Controller* harus mampu menerima sebuah paket yang berasal dari switch.
2. *Controller* harus mampu menerima informasi berupa paket dari *Web Server* yang dikirimkan oleh sebuah agen Psutil.
3. *Controller* harus mampu melakukan perhitungan menggunakan algoritme *Fuzzy* dan mengarahkan paket ke server yang memiliki kondisi beban lebih ringan.
4. Controller harus mampu menjalankan modul Pox *Versi Carp* pada *load balancing*.

4.2.1.2 SDN *Switch*

1. Switch harus mampu menerima informasi atau paket yang dikirim dari *client.*
2. Switch harus mampu mengirimkan informasi atau paket yang datang dari client ke controller (*packet in message*).
3. Switch harus mampu menerima pesan dari *controller* mengenai perintah yang harus dilakukan terhadap suatu paket (*packet out message*).
4. Switch harus dapat berjalan pada *openvswitch versi* 2.8.1.

4.2.1.3 Web Server

1. *Web server* harus mampu menyediakan layanan *website.*
2. *web server* harus mampu menerima suatu *request* dan merespon *request* tersebut.

4.2.1.4 Agen Psutils

1. Agen *Psutils* harus mampu mendapatkan data server berupa infromasi penggunaan sumber daya dari setiap server.
2. Agen *Psutils* harus mampu mengirimkan data server berupa infromasi penggunaan sumber daya dari setiap server ke *Controller*.

4.2.2 Kebutuhan Non-Fungsional

Dalam kebutuhan non fungsional terdapat batasan dari sistem yang dibangun. Berikut merupakan kebutuhan non-fungsional dari sistem ini:

1. Sistem hanya mampu berjalan pada sistem operasi Linux Ubuntu 16.04.

2. Sistem hanya menggunakan 3 server yang menggunakan layanan *website* yang sama.
3. Sistem bisa dijalankan hanya pada jaringan lokal.
4. Sistem hanya menggunakan alamat IP *virtual* tidak mengunakan IP publik atau domain sebagai IP *Web server*.

4.2.3 Kebutuhan Lingkungan Kerja Sistem

Pada tahap ini, peneliti akan menjelaskan tentang kebutuhan yang diperlukan oleh sistem agar sistem dapat berjalan. Kebutuhan yang dimaksud adalah kebutuhan perangkat lunak dan kebutuhan perangkat keras.

4.2.3.1 Kebutuhan Perangkat Lunak

1. Sistem operasi yang digunakan pada penelitian ini adalah Linux Ubuntu 16.04
2. *Virtual* switch yang digunakan adalah *open vSwitch* dengan versi 2.8.1
3. Httperf yang digunakan untuk melakukan pengujian tehadap kinerja *load balancing* server.

4.2.3.2 Kebutuhan Perangkat Keras

Dalam tahap ini, komponen yang beserta kemampuan dari perangkat terlihat di Tabel 4.1.

Tabel 4. 1 Spesifikasi Perangkat Keras

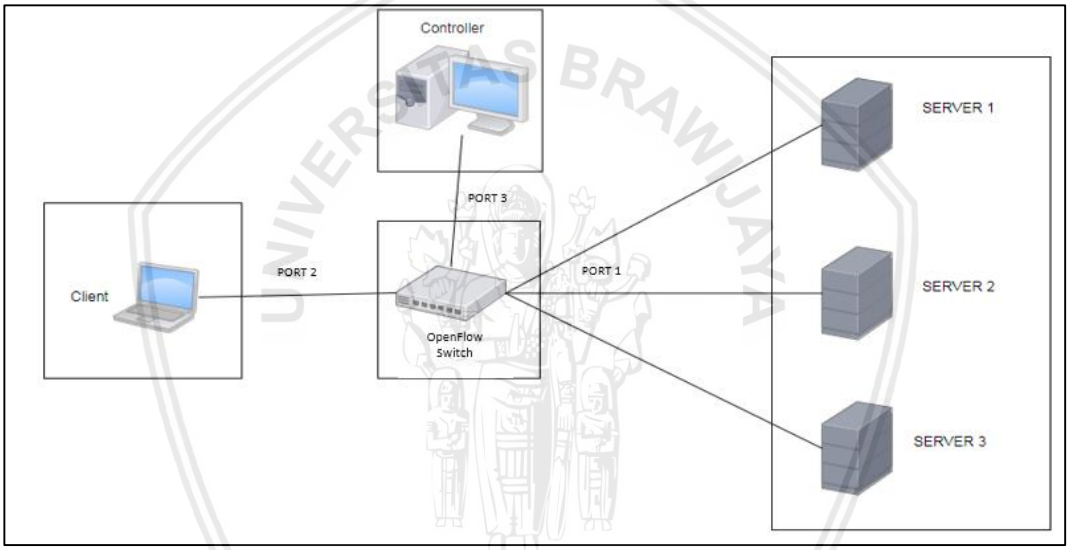
Komponen	Prosesor	Ram	jenis
SDN Switch	Intel Celeron	4 Giga Byte	Laptop
Server (*Virtual Machine*)	Intel Core i5	16 Giga Byte	Komputer
Controller	Intel Core i7	4 Giga Byte	Laptop
Client	Intel Core i5	4 Giga Byte	Laptop

BAB 5 PERANCANGAN

Dalam tahap ini merupakan perancangan sistem untuk mendukung tahap implementasi pada bab implementasi. Adapun proses perancangan yaitu: perancangan arsitektur SDN, *load balancing*, algoritme *Fuzzy* dan algoritme pada agen *Psutil*.

5.1 Perancangan Arsitektur SDN

Dalam tahap perancangan arsitektur SDN, sistem akan dirancang berdasarkan topologi di Gambar 5.1. Adapun topologi yang dirancang dengan menggunakan 3 buah server sebagai *Web Server*, 1 buah Controller, 1 buah *OpenFlow Switch* dan 1 buah *client*. *Controlle*r, *Client* dan Server akan terhubung secara langsung ke *OpenFlow Switch.*



Gambar 5. 1 Topologi SDN

5.1.1 POX Controller

Controller yang akan digunakan merupakan *Controller* POX, yang menggunakan *library* yang tersedia pada Python untuk dapat diimplementasikan di arsitektur SDN. Pada *library* ini juga akan dirancang mekanisme pembagian beban menggunakan algoritme *Fuzzy*. Pada saat yang bersamaan juga library ini akan mengatur aktifitas yang terdapat pada jaringan.

5.1.2 Open vSwitch

Open flow switch dapat dijalankan, apabila menggunakan sebuah aplikasi *open vSwitch* untuk dapat menghubungkan perangkat ke switch. Aplikasi tersebut, berfungsi sebagai SDN switch yang terpasang pada laptop.

5.1.3 Flask

Untuk membuat *Web Server,* dibutuhkan Flask yang merupakan *library* pada Python. Library tersebut, dapat mendukung pada proses pengujian sistem *load balancing*.

5.1.4 Pengalamatan IP Address Perangkat

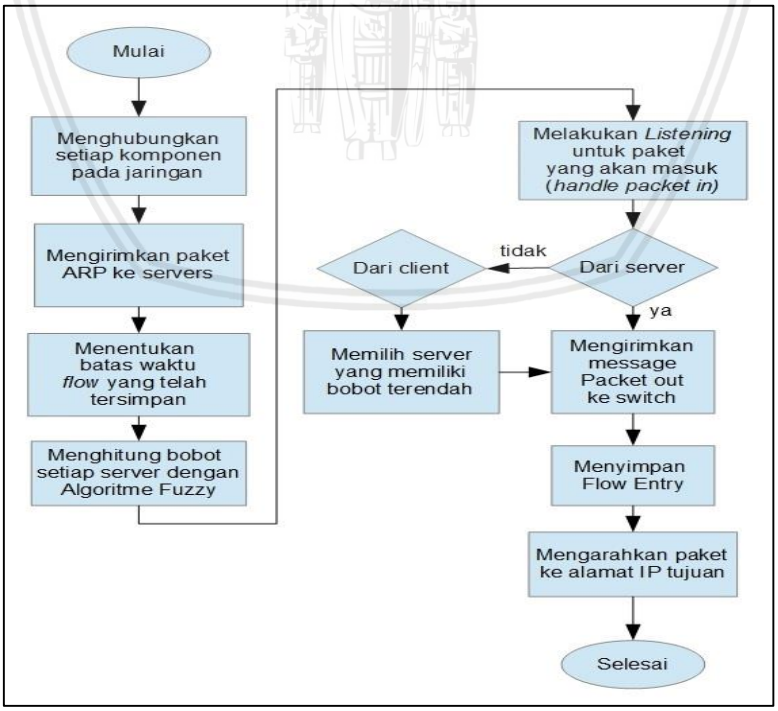
Pada bagian ini agar seluruh server dapat dijalankan, diperlukan pengalokasian IP. Adapun alamat IP pada setiap perangkat yang digunakan terlihat pada Tabel 5.1:

Tabel 5. 1 Pengalamatan Internet Protocol Address

Perangkat	Internet Protocol Address	Media Access Control Address
Web Server (S1)	10.0.0.11	00:00:00:00:00:01
Web Server (S2)	10.0.0.12	00:00:00:00:00:02
Web Server (S3)	10.0.0.13	00:00:00:00:00:03
Controller	10.0.0.20	00:00:00:00:00:04
Klien	10.0.0.10	00:00:00:00:00:05

5.2 Perancangan Load Balancing

5.2.1 Alur Sistem Load Balancing

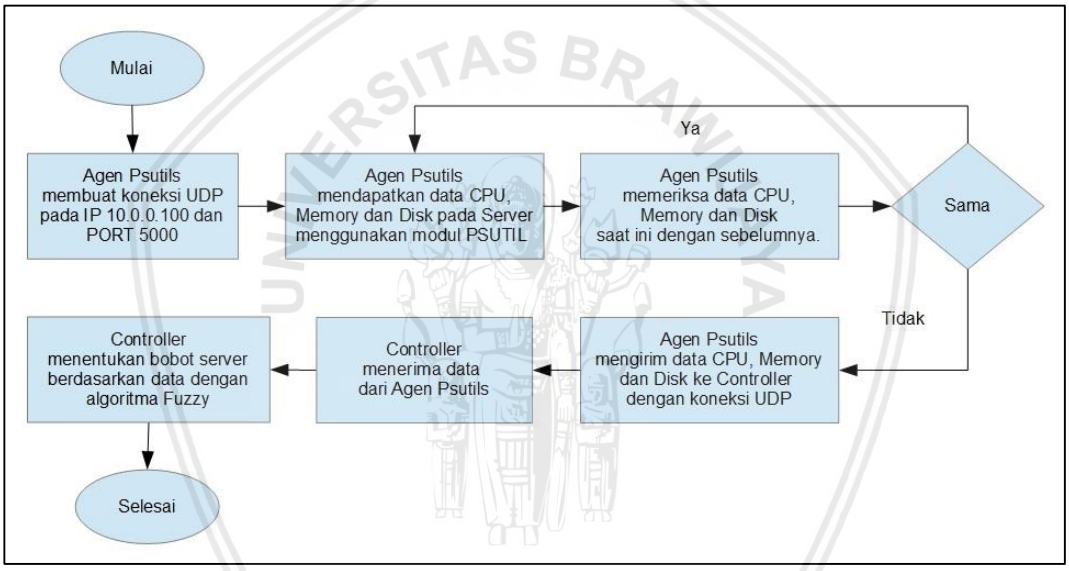


Gambar 5. 2 Flowchart Sistem *Load Balancing*

Berdasarkan Gambar 5.2 adalah proses terjadinya *load balancing.* Setelah terhubungnya komponen pada jaringan, maka *Controller* mengirimkan paket ARP pada setiap server. Kemudian menentukan batas waktu *flow* pada *Controller* dan *switch* dan menghitung bobot setiap server menggunakan algoritme *Fuzzy.* selanjutnya *controller* melakukan *listening* untuk paket yang akan masuk (*handle packet in).* jika paket terkirim dari klien, maka paket diteruskan ke server yang memiliki bobot terendah oleh *Controller*. Sedangkan jika paket terkirim dari server, maka pesan paket out dikirimkan ke switch. Pesan tersebut, berasal dari *Controller* yang disimpan pada *flow entry* untuk diarahkan ke alamat IP tujuan.

5.2.2 Mekanisme Penentuan Bobot Server

Pada proses ini, setiap server akan ditanamkan sebuah agen. Agen tersebut berfungsi untuk mengirimkan data terkait sumber daya *Web Server* seperti: CPU *usage*, *memory usage,* dan disk secara *real time*.



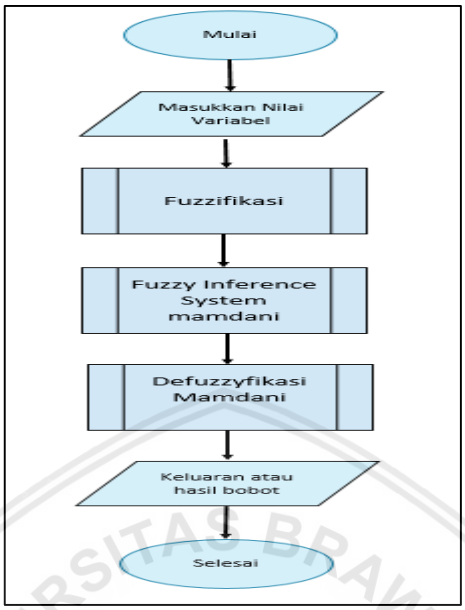
Gambar 5. 3 Flowchart Mekanisme Penentuan Bobot Server

Berdasarkan pada Gambar 5.3 merupakan mekanisme untuk pembobotan server menggunakan agen *Psutil*. Langkah pertama terhubungnya server pada Controller melalui socket UDP. Setelah terhubung, agen tersebut akan mengambil dan menyimpan data berupa data kondisi server. Kemudian data yang sudah didapatkan akan dibandingkan dengan data yang baru. Apabila keadaan server baru berbeda dengan server yang lama, maka data baru akan disimpan di *Controller.* Dengan adanya mekanisme ini, agen tidak akan mengirimkan data server yang sama dan membantu mengurangi beban jaringan.

5.3 Perancangan Algoritme *Fuzzy*

Didalam tahap perancangan ini, metode fuzzy digunakan untuk dapat membagi beban berdasarkan sumber daya server terendah pada sistem *load balancing*. Pendistribusian tersebut berdasarkan: disk, *memory* dan CPU. Dari

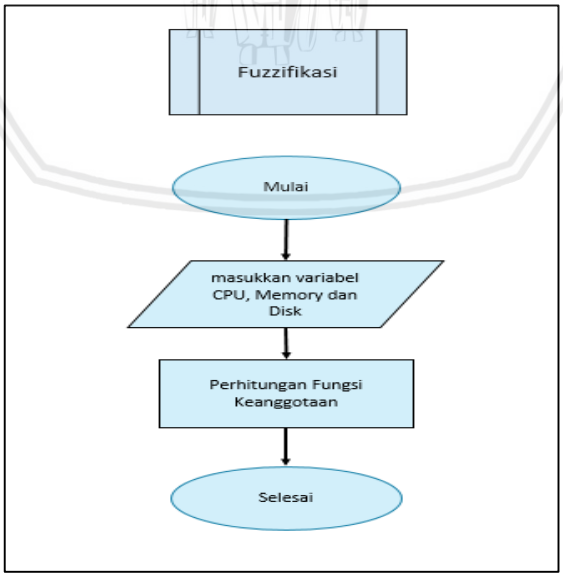
ketiga variabel tersebut, akan dijadikan variabel input untuk penghitungan algoritme *Fuzzy*.



Gambar 5. 4 Alur FIS Mamdani

Berdasarkan Gambar 5.4 adalah tahapan-tahapan pada proses FIS Mamdani. Diawali dengan pendefinisian variabel input dan juga output. Setelah itu, melakukan fuzzifikasi untuk mencari nilai output. Kemudian masuk pada tahap *Fuzzy inference system* Mamdani agar dapat melakukan tahap *defuzzifikasi* untuk memperoleh bobot server.

5.3.1 Fuzzifikasi Mamdani



Gambar 5. 5 Flowchart Fuzzifikasi

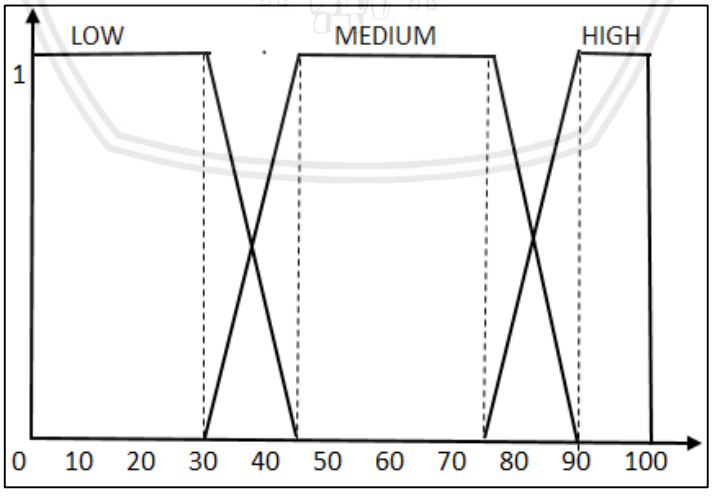
Pada titik ini, tahap fuzzifikasi mamdani yang bertujuan mengubah bentuk fuzzy dalam bentuk variabel linguistik dengan fungsi keanggotaan

tertentu. Selain itu, tahap fuzzifikasi memiliki empat variabel yang dikonversi menjadi angka fuzzy berupa tiga variabel sebagai input dan satu variabel sebagai outputnya. Untuk variabel input adalah CPU, disk dan memori. Sedangkan untuk variabel outputnya adalah bobot server. Kemudian untuk alur fuzzifikasi terlihat pada Gambar 5.5.

Tabel 5. 2 Variabel Input Linguistik Fuzzy

Variabel *Fuzzy*	Variable Linguistik	jarak	Simbol satuan
Prosesor	LOW	0 – 45	Persen (%)
	MEDIUM	30 – 90	
	HIGH	80 -100	
RAM	LOW	0 – 45	Persen (%)
	MEDIUM	30 – 90	
	HIGH	80 – 100	
Disk	LOW	0 – 45	Persen (%)
	MEDIUM	30 – 90	
	HIGH	80 – 100	

Pada peneltian ini, terdapat tiga himpunan variabel input yakni: *low*, *medium* dan *high*, hal tersebut mengacu atau berdasarkan penelitian oleh sebuah perusahaan di bidang teknologi informasi (Citrix, 2016). Sedangkan pada Tabel 5.3 merupakan variabel output yang memiliki lima himpunan yakni: *very high, high, medium, very low* dan *low.* Kemudian variabel *Fuzzy* input menentukan derajat keanggotaannya menggunakan persamaan 5.1, 5.2 dan 5.3.



Gambar 5. 6 Nilai Drajat Keanggotaan Variabel Input

Dalam Gambar 5.6 menjelaskan nilai dari drajat keanggotaan untuk variable input yang tebagi kedalam 3 kategori yaitu: *high, medium* dan *low.* Untuk kategori low grafik keanggotaan digambarkan dengan bentuk *left shoulder*

trapezoid dan kategori medium grafik keanggotaan digambarkan dengan bentuk *trapezoid*, serta kategori *high* grafik keanggotaan digambarkan dengan bentuk *right shoulder trapezoid*.

$$\mu(x)\ Low = \begin{cases} 1; & x \leq 30 \\ \dfrac{45 - x}{45 - 30} & 30 \leq x \leq 45 \\ 0; & x \geq 45 \end{cases} \qquad (5.1)$$

$$\mu(x)\ Medium = \begin{cases} 0; & x \leq 30\ atau\ x \geq 90 \\ \dfrac{x - 30}{45 - 30}; & 30 \leq x \leq 45 \\ 1; & 45 \leq x \leq 90 \\ \dfrac{90 - x}{90 - 80}; & 80 \leq x \leq 90 \end{cases} \qquad (5.2)$$

$$\mu(x)\ High = \begin{cases} 0; & x \leq 80 \\ \dfrac{x - 80}{90 - 80} & 80 \leq x \leq 90 \\ 1; & x \geq 90 \end{cases} \qquad (5.3)$$

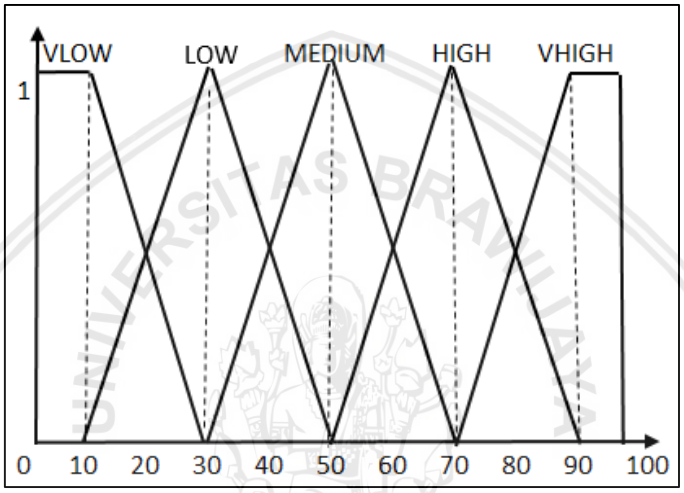
Pada persamaan 5.1 merupakan cara untuk menghitung drajat keanggotaan dari himpunan Fuzzy low dari setiap variable input. Jika setiap variable input tidak mencapai nilai 30, maka nilai dari drajat keanggotaan tersebut adalah 1 dan jika setiap variable input melebihi nillai 30 dan tidak mencapai nilai 45, maka drajat keanggotaanya sama pada persamaan diatas. Selanjutnya jika setiap variable input melebihi nilai 45 dan tidak mencapai nilai 80, maka nilai dari drajat keanggotaan nilainya 0.

pada persamaan 5.2 merupakan cara untuk menghitung drajat keanggotaan dari himpunan Fuzzy medium dari setiap variable input. Jika setiap variable input melebihi nilai 30 dan tidak mencapai 90, maka nilainya dari drajat keanggotaan adalah 0 dan jika setiap variable input melebihi nilai 30 dan tidak mencapai nilai 45, maka nilainya dari drajat keanggotaan adalah sama pada persamaan diatas. Selanjutnya jika setiap variable input melebihi nilai 45, maka nilainya dari drajat keanggotaan adalah 1, lalu jika setiap variable input melebihi nilai 75 dan tidak mencapai nilai 90, maka nilainya dari drajat keanggotaa adalah sama pada persamaan diatas.

Pada persamaan 5.3 merupakan cara untuk menghitung drajat keanggotaan dari himpunan Fuzzy *high* dari setiap variable input. Jika setiap variable input tidak mencapai nilai 75, maka nilainya dari drajat keanggotaan adalah 0 dan jika setiap variable input melebihi nilai 75 dan tidak mencapai 90, maka nilainya dari drajat keanggotaan adalah sama pada persamaan diatas. Selanjutnya jika setiap variable input bernilai lebih dari 90, maka nilai dari drajat keanggotaannya adalah 1.

Tabel 5. 3 Variabel Output Linguistik Fuzzy

Variabel Fuzzy	Variabel Linguistik	Jarak	Simbol satuan
Bobot Server	VERY LOW	0 – 30	Persen (%)
	LOW	10 – 50	
	MEDIUM	30 – 70	
	HIGH	50 – 90	
	VERY HIGH	80 – 100	



Gambar 5. 7 Nilai drajat Keanggotaan Variabel Bobot Server

Berdasarkan Gambar 5.7 terdapat lima fungsi keanggotaan. Dalam kategori sangat rendah (VLOW), drajat keanggotaannya berbentuk *left shoulder trapezoid*. Kemudian dalam kategori rendah drajat keanggotaannya dalam bentuk *triangel*. Berikutnya di kategori tengah dengan drajat keanggotaan dalam bentuk *triangel*. Kemudian dalam kategori tinggi (HIGH) drajat keanggotaannya dalam bentuk *triangel*. Kategori terakhir sangat tinggi (VHIGH) dalam grafik keanggotaannya *right shoulder trapezoid.*

5.3.2 Aplikasi Fungsi Implikasi

Pada tahap ini, setiap variabel input memiliki kondisi yang berbeda-beda. Oleh karena itu, dibuat sebuah mekanisme aturan fungsi implikasi *Fuzzy* yaitu “IF” yang akan menghasilkan output "THEN" yang terlihat di tabel berikut.

Tabel 5. 4 Rules Variabel *Fuzzy*

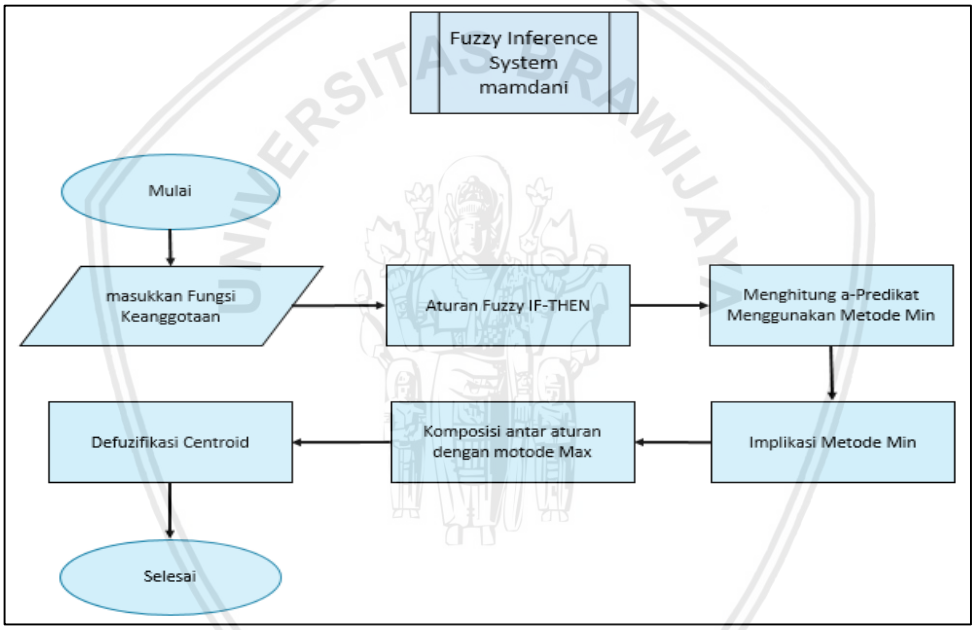
Aturan:	CPU	Memory	Disk	Bobot Server
Aturan 1	Low	Low	Low	Very low
Aturan 2	Low	Low	Medium	Very low

Aturan 3	Low	Low	High	Low
Aturan 4	Low	Medium	Low	Very low
Aturan 5	Low	Medium	Medium	Medium
Aturan 6	Low	Medium	High	Medium
Aturan 7	Low	High	Low	Low
Aturan 8	Low	High	Medium	Medium
Aturan 9	Low	High	High	High
Aturan 10	Medium	Low	Low	Medium
Aturan 11	Medium	Low	Medium	Medium
Aturan 12	Medium	Low	High	Medium
Aturan 13	Medium	Medium	Low	Medium
Aturan 14	Medium	Medium	Medium	Medium
Aturan 15	Medium	Medium	High	High
Aturan 16	Medium	High	Low	Medium
Aturan 17	Medium	High	Medium	High
Aturan 18	Medium	High	High	Very high
Aturan 19	High	Low	Low	Low
Aturan 20	High	Low	Medium	Medium
Aturan 21	High	Low	High	High
Aturan 22	High	Medium	Low	Medium
Aturan 23	High	Medium	Medium	High
Aturan 24	High	Medium	High	Very high
Aturan 25	High	High	Low	Very high
Aturan 26	High	High	Medium	Very high
Aturan 27	High	High	High	Very high

Dalam Tabel 5.4 terdapat peraturan Fuzzy yang telah dibuat menggunakan model IF-THEN berdasarkan set variabel input. Kemudian, dari peraturan fuzzy menjadi dasar proses fungsi implikasi. Metode yang digunakan pada proses tersebut yaitu MIN.

5.3.3 Sistem FIS Mamdani

Fuzzy Inferensi Sistem (FIS) Mamdani menggunakan metode Min-Max. Untuk mengetahui nilai Crips sebagai output, diawali dengan variabel input dan output yang dikonversikan menjadi himpunan Fuzzy dengan aturan Fuzzy yang dibuat. Adapun metode min digunakan untuk menghasilkan a-predikat pada proses FIS Mamdani dalam fungsi implikasi. Setelah hasil fungsi implikasi min telah diperoleh, untuk mendapat hasil dari daerah fuzzy menggunakan metode Max pada proses komposisi aturan. Berikut adalah beberapa langkah FIS mamdani ditunjukkan dalam Gambar 5.8.



Gambar 5. 8 Flowchart FIS Mamdani

Berdasarkan Gambar 5.8, peneliti akan mencontohkan proses perhitungan *Fuzzy* Inference System Mamdani. Adapun contohnya untuk setiap variable input bernilai 35%. Kemudian melakukan proses fuzzifikasi yang menghasilkan variabel *Fuzzy* input dengan nilai drajat keanggotaan himpunan Fuzzy *low* 0,75, dan nilai drajat keanggotaan himpunan Fuzzy medium adalah 0,25. Kemudian nilai drajat keanggotaan himpunan Fuzzy *high* adalah 0. Selanjutnya lakukanlah proses inferensi *Fuzzy* dengan cara persamaan berikut.

Aturan [1]: jika CPU *low* dan *Memory low* dan Disk *low* maka bobot *very low*

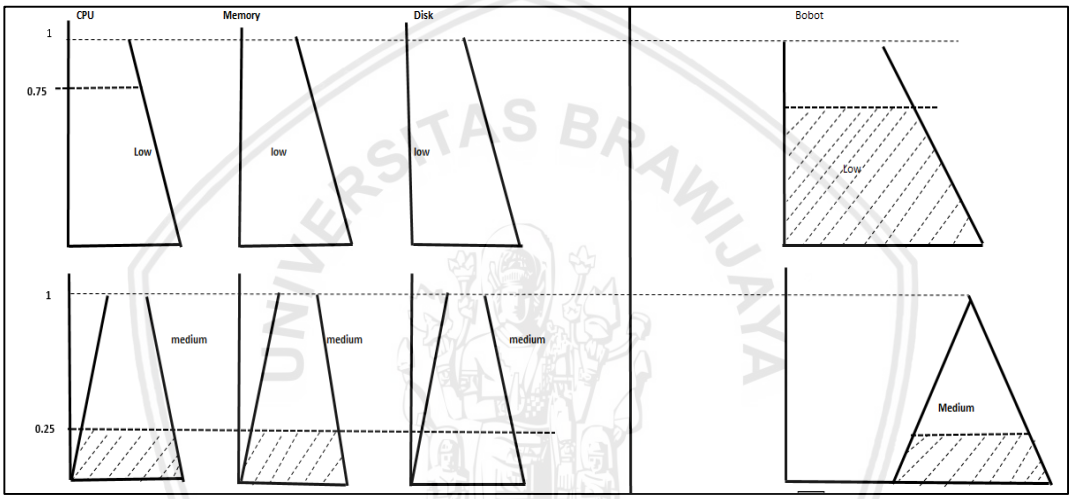
$$a - predikat_1 = \min(\mu Cpu\ \text{low}, \mu Memory\ \text{low}, \mu Disk\ \text{low})$$
$$= \min(0.75, 0.75, 0.75) \quad (5.4)$$

$$= 0.75$$

Aturan[5]: IF CPU medium AND *Memory* medium AND Disk medium THEN bobot medium

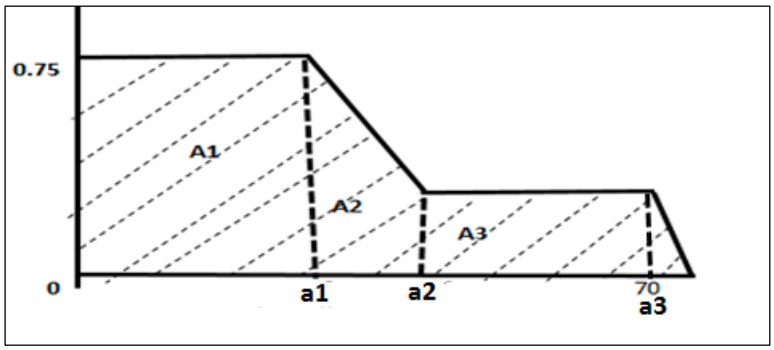
$$a - predikat_5 = \min(\mu Cpu\ medium, \mu Memory\ medium, \mu Disk\ medium)$$
$$= \min(0.25,0.25,0.25) \quad (5.5)$$
$$= 0.25$$

Setelah nilai dari predikat a diketahui dalam semua aturan, langkah selanjutnya merupakan implikasi dan komposisi aturan. Langkah tersebut menggunakan metode Min-Max yang akan menghasilkan daerah hasil fuzzy seperti yang ditunjukkan di bawah ini.



Gambar 5. 9 FIS Mamdani Metode Min-Max

Deskripsi umum proses inferensi akan menggunakan metode FIS mamdani ditunjukkan pada Gambar 5.9. Proses mencari predikat alfa menggunakan metode min untuk setiap aturan. Kemudian proses implikasi komposisi aturan yang yang akan menggunakan metode min-max untuk mendapatkan area hasil Fuzzy, proses berikutnya memasuki tahap defuzzifikasi. Metode yang digunakan pada proses defuzzifikasi adalah metode Centroid. Metode tersebut akan membagi daerah hasil fuzzy ke beberapa bagian. Sebelum itu, diperlukan untuk menghitung batas-batas yang ada pada daerah tersebut. Pada Gambar 5.10 adalah proses pencarian daerah yang dihasilkan dari metode Centorid. Daerah hasil Fuzzy terbagi menjadi tiga bagian berupa A1, A2 dan A3 yang memiliki batas zona dari setiap bagian yakni (a1), (a2) dan (a3) dengan persamaan 5.6, 5.7 dan 5.8.



Gambar 5. 10 area Hasil *Fuzzy*

$$a1 = 50 - 0.75(50 - 30)$$
$$= 50 - 15 \qquad (5.6)$$
$$= 35$$

$$a2 = 50 - 0.25(50 - 30)$$
$$= 50 - 5 \qquad (5.7)$$
$$= 45$$

$$a3 = 75 - 0.25(70 - 50)$$
$$= 75 - 5 \qquad (5.8)$$
$$= 70$$

Setelah mendapatkan nilai batas area a1, a2 dan a3 yang didapatkan dari bagian hasil daerah fuzzy yang terbagi pada 3 bagian A1, A2 dan A3. Maka menghasilkan persamaan 5.9 yang digunakan untuk fungsi keanggotaan ditentukan sebagai berikut untuk darea hasil fuzzy.

$$\mu(z) = \begin{cases} 0.75; & z \leq 35 \\ \dfrac{50 - z}{50 - 30}; & 35 \leq z \leq 45 \\ 0.25; & 45 \leq z \leq 70 \\ \dfrac{70 - z}{70 - 50}; & 70 \leq z \leq 75 \end{cases} \qquad (5.9)$$

Berdasarkan persamaan yang sama seperti di atas, untuk mendapatkan nilai output berupa nilai crisp value diperoleh dengan cara persamaan 5.10.

$$z^* = \frac{\Sigma_{j=1}^{n}\, zj * (zj)}{\Sigma_{j=1}^{n}\, \mu\,(zj)} \qquad (5.10)$$

Langkah selanjutnya merupakan proses perhitungan untuk mengkonversi dari angka fuzzy ke nilai crisp seperti dibawah ini.

$$z^* = \frac{(20 + 25 + 30) * 0.75 + (35 + 40 + 45) * 0{,}25 + (50 + 60 + 70) * 0.25}{0.75 * 3 + 0{,}25 * 3 + 0.25 * 3}$$

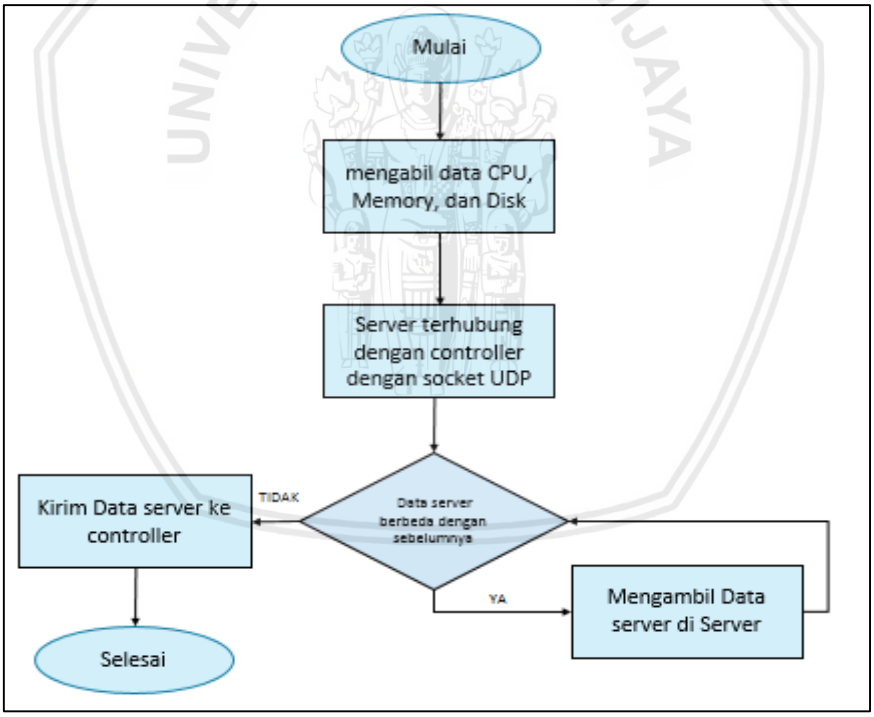
$$z^* = \frac{131.25}{3{,}75}$$

$$z^* = 35$$

Telihat pada baris pertama, angkat 20, 25, 30, 35, 40, 45, 50, 60 dan 70 merupakan angka sembarang (arbitrer) yang diambil dari nilai alfa predikat pada proses komposisi aturan. Setelah melakukan proses perhitungan, dapat diambil kesimpulan bahwa apabila variabel input memiliki nilai 35%, berarti nilai output yang dihasilkan 35% sebagai bobot server.

5.4 Perancangan Algoritme pada *Agen Psutils*

Pada bagian ini, fungsi agen akan mengirim informasi berupa penggunaan sumber daya server melalui socket UDP.



Gambar 5. 11 Flowchart *Agen Psutils*

Berdasarkan dalam Gambar 5.11 diawali dengan agen mendapatkan informasi penggunaan sumber daya server agar disimpan dalam *controller*. Dalam proses perulangan informasi data server, pemeriksaan data server baru diambil dengan data server sebelumnya bertujuan untuk mendapatkan informasi penggunaan sumber daya server terbaru dan dikirim ke *controller*.

BAB 6 IMPLEMENTASI

Bagian ini menjelaskan tentang penerapan sistem *load balancing* menggunakan algoritme *Fuzzy*. Adapun tahapan untuk menerapkan sistem tersebut yaitu konfigurasi perangkat yang meliputi konfigurasi server, *client*, SDN Switch dan SDN *Controller*. Kemudian dilanjutkan dengan implementasi Arsitektus SDN, *Load Balancing*, Algoritme *Fuzzy* dan Algoritme pada Agen *Psutils*.

6.1 Konfigurasi Perangkat

Pada titik ini, seluruh perangkat yang akan digunakan oleh penelitian akan dikonfigurasikan sebelumnya supaya berfungsi dengan benar. Perangkat tersebut yaitu: *Controller,* server, klien dan *Switch*. Berikut ini penjelasan lebih rinci mengenai pengaturan perangkat.

6.1.1 Konfigurasi Server

Pada penelitian ini menggunakan tiga server virtual pada perangkat lunak *Virtual Box*. Selanjutnya melakukan pengaturan pada server dan menentukan partisi serta alamat IP setiap server.

1. Pengaturan Server

Setiap *server* memiliki jumlah core dan *memory* yang berbeda.Berikut ini merupakan pembagian core dan *memory* pada tiap *server.*

- *S1 :* Jumlah *core 4* dan *memory* 4
- *S2 :* Jumlah *core 2* dan *memory* 2
- *S3:* Jumlah *core 1* dan *memory* 1

2. Pengalamatan IP Server

Setelah menentukan jumlah core yang ditentukan, kemudian memberi alamat IP tiap *virtual server*. Alamat IP di Sistem Operasi Ubuntu dapat diatur menggunakan perintah ini:

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

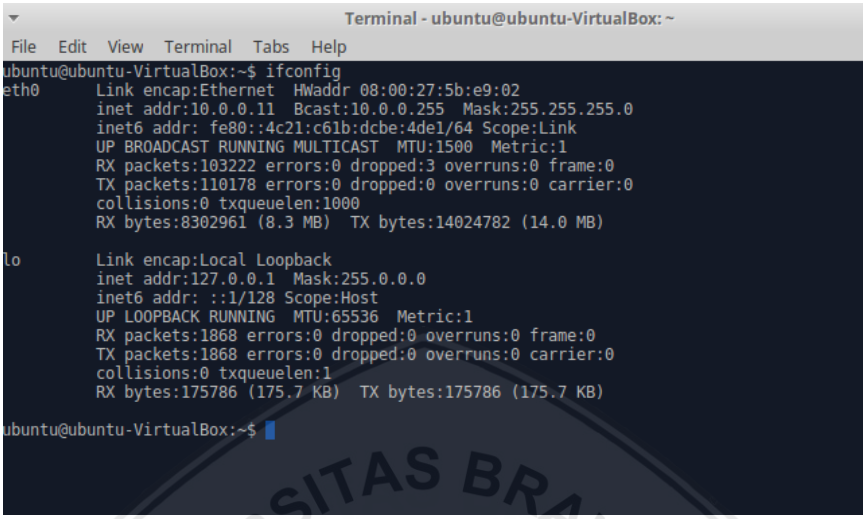
Alamat IP diterapkan untuk masing-masing server sebagai berikut.

```
Auto eth0
iface eth0 inet static
    address 10.10.0.11
    netmask 255.255.255.0
gateway 10.0.0.0
```

Setelah konfigurasi, alamat IP terlihat dengan syntax dibawah ini.

```
$ sudo ifconfig-a
```

Setelah melakukan perintah diatas maka ditampilkan antarmuka server dapat dilihat di Gambar 6.1 berikut.



Gambar 6. 1 Antarmuka Server

Tahapan ini diterapkan pada semua server menggunakan alamat IP dibawah ini .

- *Internet* Protocol *address* 10.0.0.11 sebagai server 1
- *Internet Protocol address* 10.0.0.12 sebagai server 2
- *Internet* Protocol *address* 10.0.0.13 sebagai server 3

6.1.2 Konfigurasi Client

Pada tahap konfigurasi klien dibutuhkan sebuah aplikasi untuk melakukan tes adalah httperf. Kemudian, Klien dapat mengirim permintaan menggunakan aplikasi tersebut sesuai kebutuhan pengujian. Selanjutnya, alamat IP klien dialihkan sehingga terhubung pada Open vSwitch. Adapun perintah tersebut adalah dengan cara dibawah.

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

Proses selanjutnya merupakan konfigurasikan *internet protocol address* untuk masing-masing klien dengan cara berikut.

```
autoieth0
iface eth0iinetistatic
    addressy10.0.0.20
    netmasky255.255.255.0
    gatewayy10.0.0.0
```

Kemudian, setelah proses pengaturan klien, klien dapat membuat permintaan dengan perintah dibawah.

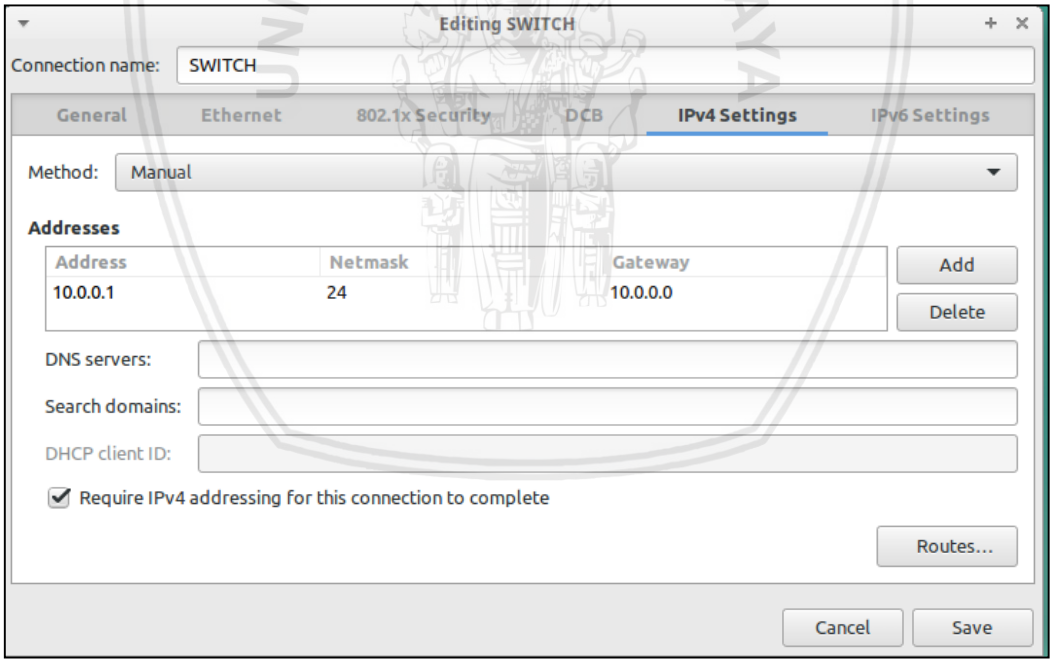
```
$ httperf –server [ip service] --port 80 --num-conns a --rate b
```

Berdasarkan perintah diatas, klien dapat membuat permintaan yang berdasarkan dengan persyaratan pengujian yang dibutuhkan.

6.1.3 Konfigurasi SDN Switch

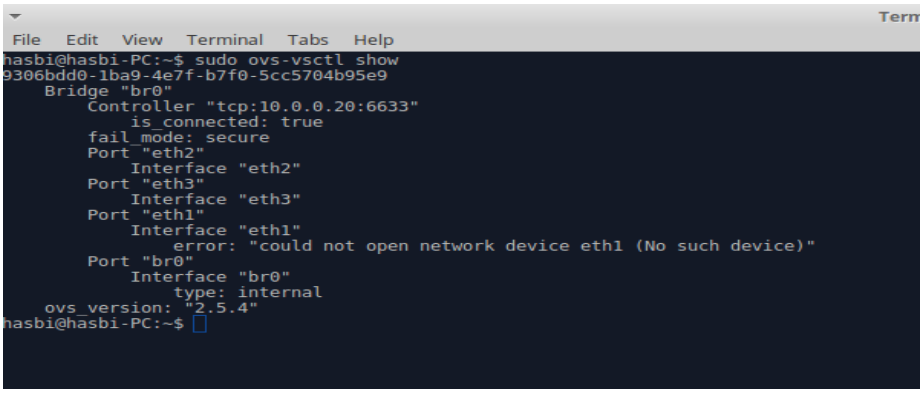
Switch yang dipakai pada penelitian adalah switch virtual yaitu Open vSwitch. Pertama yang harus dilakukan yaitu membuat *bridge* baru yang diberi nama br0. Bridge akan bekerja meneruskan paket berdasarkan flow tabel yang diberikan. Kemudian setelah membuat *bridge* baru selesai, dilanjutkan dengan menambahkan port-port yang nantinya akan dipakai untuk menghubungkan tiga server, *Controller* dan klien. Untuk dapat memisahkan dan menghubungkan Switch dengan *Controller,* harus dilakukan konfigurasi terlebih dahulu pada Switch. Setelah konfigurasi berhasil dilakukan, switch dapat dihubungkan dengan *Controller* dan mendefinisikan *IP Controller* yang akan digunakan pada Switch.

Setelah Proses konfigurasi bridge selesai, kita dapat mebambahkan port yang akan dipakai oleh switch untuk menerima paket. Port tersebut harus memiliki *interface,* sehingga host dan Switch dapat bertukar data. Konfigurasi alamat IP terlihat pada Gambar 6.2



Gambar 6. 2 Alamat IP *Switch.*

Berdasarkan Gambar 6.3 dibawah, menjelaskan bahwa Controller sudah dalam keadaan terhubung dengan Switch dan terdapat informasi Controller "tcp:10.0.0.20:6633" *is_connected: true.* Kemudian ada tiga port yang sudah siap digunakan untuk menghubungkan server, klien dan *Controller*



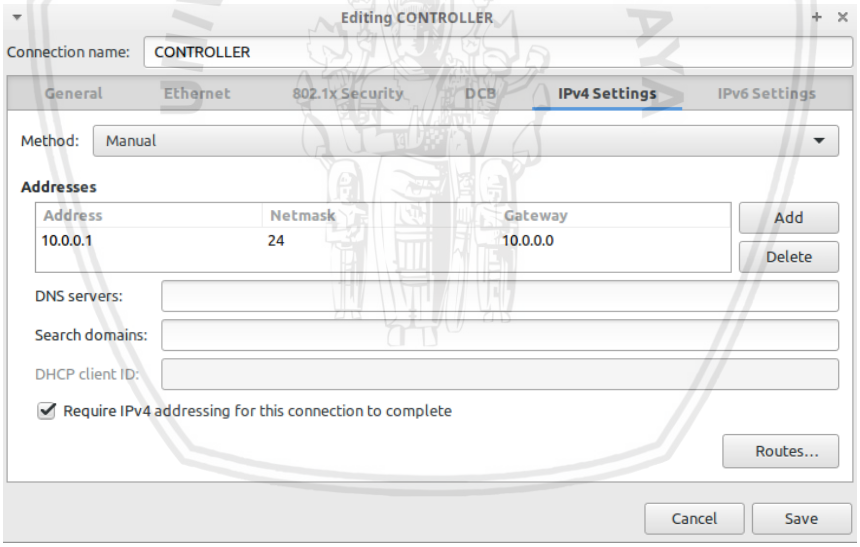
Gambar 6. 3 Pengaturan *Open vSwitch*

6.1.4 Konfigurasi SDN Controller

Untuk dapat menjalankan SDN Controller, dengan perintah yang ada sebagai berikut ini.

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

Setelah itu dilakukan konfigurasi *internet protocol address* seperti Gambar 6.4.



Gambar 6. 4 Alamat IP *Controller*

6.2 Implementasi Arsitektur SDN

6.2.1 Instalasi POX Controller

Untuk menginstal POX Controller, kita dapat mengkloning repositori noxrepo, yang terletak di http://github.com/noxrepo/pox. Setelah itu, Controller POX dapat dijalankan dari direktori POX yang disimpan di Controller.

6.2.2 Instalasi Open vSwitch

Untuk dapat menjalankan SDN Switch, sebenarnya dibutuhkan perangkat lunak yang dapat bertindak seperti switch fisik. Untuk itu kami memakai Open vSwitch untuk menginstal sistem pada lingkungan Instalasi Server Web.

6.2.3 Instalasi *Web Server*

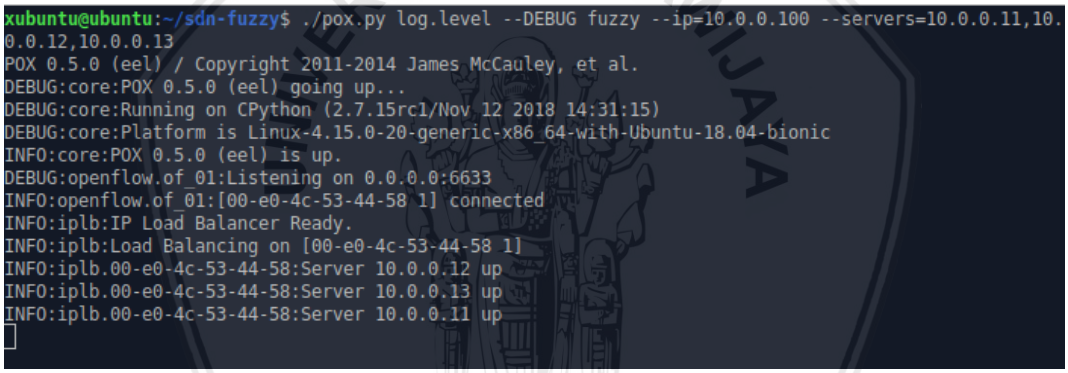
Untuk menginstal *Web Server,* dibutuhkan Flask yang merupakan *library* pada Python. Namun, untuk menginstall flask juga dibutuhkan python-pip

6.3 Implementasi Load balancing

Hal pertama untuk dapat menerapkan load balancing di server web pada SDN merupakan membuat topologi. Topologi tersebut, sudah dijelaskan pada tahap desain arsitektur SDN. Berikut perintah untuk menjalankan *Controller Pox.*

```
$ sudo python pox.py log.level –DEBUG Fuzzy --ip=10.0.0.100 --servers=10.0.0.11,10.0.0.12,10.0.0.13
```

Apabila prosesnya dapat berjalan, maka seperti Gambar 6.5 dibawah ini.



Gambar 6. 5 Tampilan Pox Controller Dijalankan

Bedasarkan Gambar 6.5 merupakan tampilan dari *Pox Controller. Controller* tersebut menggunakan algoritme *Fuzzy* dan juga agen Psutil. Opsi dari -ip = 10.0.0.100 adalah *IP Address Web server* yang dapat digunakan oleh klien. Kemudian opsi dari –servers = 10.0.0.11,10.0.0.12,10.0.013 adalah *IP address* dari setiap server yang terhubung pada *Controller.* Selanjutnya pada tamplin pox Controller terdapat informasi mengenai kondisi dari setiap server yang terhubung yang dinyatakan dengan *IP address* masing-masing server.

Berikut ini merupakan cara menjalankan server_agent.py yang merupakan perintah untuk menjalankan agen *Psutil.*

```
$ sudo python server.py && sudo python server_agent.py
```

Apabila agen dapat berjalan, maka dapat dilihat dalam Gambar 6.7.

Gambar 6. 6 Interface Terminal pada server 1

Gambar 6. 7 Interface Terminal Ketika datang *Request*

Dari demonstrasi diatas, dapat dinyatakan bahwa sistem *load balancing* menggunakan beberapa modul yang terdapat di setiap program, yaitu server.py dan server_agent.py. Pada program server.py ada beberapa proses untuk mengimplementasikan load balancing menggunakan perpustakaan utama, POX. Sementara server_agent.py menggunakan pustaka psutil untuk mendapatkan informasi status pada server dan soket sebagai media untuk mengirim data dengan protokol UDP.

Berikut ini adalah implementasi alur kerja dari penyeimbangan beban sistem dalam bentuk kode sumber dan penjelasan berdasarkan Aliran Sistem Penyeimbang Beban yang ditemukan pada Gambar 5.2.

Tabel 6. 1 Kode Sumber *Handle Connection Up*

```
1   def _conn_up(event):
2           global _switch_id
3           if _switch_id is None:
4               core.registerNew(iplb, event.connection, IPAddr(ip), servers)
5               _switch_id = event.dpid
6
7           if _switch_id!= event.dpid:
8               pass
9           else:
10              core.iplb.con = event.connection
11              event.connection.addListeners(core.iplb)
12      core.openflow.addListenerByName("ConnectionUp", _conn_up)
```

Proses dimulai dengan koneksi semua komponen pada jaringan seperti SDN Controller, SDN Switch, Client dan Server. Aktivitas yang ada di jaringan disebut event. Setiap event akan memiliki switch ID, maka switch ID akan didaftarkan dalam variabel di SDN Controller dan dianggap sebagai jalur komunikasi semua komponen di jaringan. Jika ada switch ID dari event yang

berbeda, aktivitas tersebut akan diabaikan. Implementasinya terlihat pada kode sumber di baris 3 sampai baris 11. Proses untuk menghubungkan komponen akan ditempatkan pada metode bernama _conn_up dengan event sebagai parameter. Kemudian akan dilanjutkan dengan mendaftarkan metode ke modul Openflow POX Listener seperti yang terlihat pada baris 14.

Tabel 6. 2 Kode Sumber Paket TCP Dari Server

```
1  if ipp.srcip in self.daftar_server:
2      kunci = ipp.srcip, ipp.dstip, tcpp.srcport, tcpp.dstport
3      entri = self.memory.get(kunci)
4      if entri is None:
5          self.log.debug("Tidak ada klien untuk %s", kunci)
6          return drop()
7      entri.refresh()
8      mac_addr, port = self.live_servers[entri.server]
9      aksi = []
10     aksi.append(of.ofp_action_dl_addr.set_src(self. mac_addr))
11     aksi.append(of.ofp_action_nw_addr.set_src(self.service_ip))
12     aksi.append(of.ofp_action_output(port= entri.client_port))
13     m = of.ofp_match.from_packet(paket, port_masuk)
14     pesan = of.ofp_flow_mod(command=of.OFPFC_ADD,
15                             idle_timeout=FLOW_IDLE_TIMEOUT,
16                             hard_timeout=of.OFP_FLOW_PERMANENT,
17                     data=event.ofp, actions= aksi,match=m)
18     self.con.send(pesan)
```

Kemudian pada Tabel 6.2 adalah proses penyaringan jenis paket TCP yang berasal dari server. Selain itu, penyimpanan aliran akan dilakukan dalam entri memori dan konstruksi paket pesan untuk Switch SDN.

Tabel 6. 3 Kode Sumber Paket TCP Dari *Client*

```
1  elif ipp.dstip == self.service_ip:
2      kunci = ipp.srcip, ipp.dstip, tcpp.srcport, tcpp.dstport
3      entri = self.memory.get(kunci)
4      if entri is None or entri.server not in self.live_servers:
5           if len(self.live_servers) == 0:
6                self.log.warn("Tidak ada servers!")
7                return drop()
8           server = self._pick_server()
9           entri = MemoryEntry(server, paket, port_masuk)
10         self.memory[entri.key1] = entri
11          self.memory[entri.key2] = entri
12     entri.refresh()
13     mac_addr, port = self.live_servers[entry.server]
14     act = []
15     act.append(of.ofp_action_dl_addr.set_dst(mac_addr))
```

```
16      act.append(of.ofp_action_nw_addr.set_dst(entri.server))
17      act.append(of.ofp_action_output(port=port))
18      m = of.ofp_match.from_packet(paket, port_masuk)
19      pesan = of.ofp_flow_mod(command=of.OFPFC_ADD,
20                              idle_timeout=FLOW_IDLE_TIMEOUT,
21                              hard_timeout=of.OFP_FLOW_PERMANENT,
22                              data=event.ofp,actions=act,match=m)
23      self.con.send(pesan)
```

Tabel 6.3 adalah proses penyaringan jenis paket TCP yang berasal dari klien. Selain itu, penyimpanan aliran akan dilakukan dalam entri memori dan konstruksi paket pesan untuk Switch SDN.

Tabel 6. 4 Kode Sumber Penentuan Server

```
1   def _pilih_server(self):
2       if len(self.data) == 0:
3           return self.live_servers.keys()[0]
4       return self.fuzz.min(self.servers_data)
```

Tabel 6.4 adalah metode yang akan menentukan server yang akan menerima permintaan melalui perhitungan Fuzzy.

6.4 Implementasi Algoritme *Fuzzy*

Proses mengimplementasikan algoritme *Fuzzy* pada load balancing pada *code* dibawah ini.

Tabel 6. 5 Kode Sumber Pendefinisian Input dan Output

```
1   self.cpu = self.input_variable('cpu')
2   self.memori = self.input_variable(memori)
3   self.disk = self.input_variable('disk')
4   self.status = self.output_variable('status')
5
6   self.in_category(self.cpu)
7   self.in_category(self.memori)
8   self.in_category(self.disk)
9   self.out_category(self.status)
```

Pada Tabel 6.5 adalah proses awal yang dimulai dengan mendefinisikan setiap variabel yang digunakan. Terdapat dua Variabel yaitu input yaitu CPU, Memori dan Disk. Kemudian variabel selanjutnya merupakan output sebagai bobot server. Setelah proses tersebut, langkah selanjutnya memasuki proses fuzzifikasi seperti kode di bawah ini.

Tabel 6. 6 Kode Sumber Fuzzifikasi

```
1   def in_category(self, in):
2           in['low'] = fuzz.trapmf(input.universe, [0, 0, 30, 45])
3           in['medium'] = fuzz.trapmf(input.universe, [30, 45, 75, 90])
```

```
4           in['high'] = fuzz.trapmf(input.universe, [75, 90, 100, 100])
5
6   def out_category(self, out):
7           out['very low'] = fuzz.trapmf(out.universe, [0, 0, 10, 30])
8           out['low'] = fuzz.trimf(out.universe, [10, 30, 50])
9           out['medium'] = fuzz.trimf(out.universe, [30, 50, 70])
10          out['high'] = fuzz.trimf(out.universe, [50, 70, 90])
11          out['very high'] = fuzz.trapmf(out.universe, [70, 90, 100, 100])
```

Kemudian pada Tabel 6.6 adalah proses fuzzifikasi variabel input dalam baris 2 hingga 4 menggunakan tingkat keanggotaan trapmf atau trapesium. Sedangkan variabel output menggunakan trimf atau derajat keanggotaan segitiga. Kemudian fungsi aturan implikasi proses pada algoritma fuzzy seperti pada kode di berikut:

Tabel 6. 7 Kode Sumber Aturan Fuzzy

```
1   def fuzz_rules(self):
2           rule1 = control.Rule(self.data_cpu['low'] & self.data_memory['low']
    & self.data_disk['low'], self.status['very low'])
3           rule2 = control.Rule(self.data_cpu['low'] & self.data_memory['low']
4   & self.data_disk['medium'], self.status['very low'])
5           rule3 = control.Rule(self.data_cpu['low'] & self.data_memory['low']
    &self.data_disk['high'], self.data_status['low'])
6           rule4 = control.Rule(self.data_cpu['low'] &
7   self.data_memory['medium'] & self.data_disk['low'], self.status['very
    low'])
8           rule5 = control.Rule(self.data_cpu['low'] &
9   self.data_memory['medium'] & self.data_disk['medium'],
    self.data_status['medium'])
10          rule6 = control.Rule(self.data_cpu['low'] &
11  self.data_memory['medium'] & self.data_disk['high'],
    self.data_status['medium'])
12          rule7 = control.Rule(self.data_cpu['low'] &
13  self.data_memory['high'] & self.data_disk['low'], self.data_status['low'])
14          rule8 = control.Rule(self.data_cpu['low'] &
15  self.data_memory['high'] & self.data_disk['medium'],
    self.data_status['medium'])
16          rule9 = control.Rule(self.data_cpu['low'] &
17  self.data_memory['high'] & self.data_disk['high'],
    self.data_status['high'])
18
19          rule10 = control.Rule(self.data_cpu['medium']  &
20  self.data_memory['low'] & self.data_disk['low'], self.status['very low'])
21          rule11 = control.Rule(self.data_cpu['medium']  &
22  self.data_memory['low'] & self.data_disk['medium'],
    self.data_status['medium'])
23          rule12 = control.Rule(self.data_cpu['medium']  &
24  self.data_memory['low'] & self.data_disk['high'],
    self.data_status['medium'])
25          rule13 = control.Rule(self.data_cpu['medium']  &
26  self.data_memory['medium'] & self.data_disk['low'],
    self.data_status['medium'])
27          rule14 = control.Rule(self.data_cpu['medium']  &
28  self.data_memory['medium'] & self.data_disk['medium'],
```

```
29  self.data_status['medium'])
30          rule15 = control.Rule(self.data_cpu['medium']  &
    self.data_memory['medium'] & self.data_disk['high'],
31  self.data_status['high'])
32          rule16 = control.Rule(self.data_cpu['medium']  &
    self.data_memory['high'] & self.data_disk['low'],
33  self.data_status['medium'])
34          rule17 = control.Rule(self.data_cpu['medium']  &
    self.data_memory['high'] & self.data_disk['medium'],
35  self.data_status['high'])
36          rule18 = control.Rule(self.data_cpu['medium']  &
    self.data_memory['high'] & self.data_disk['high'], self.status['very
37  high'])
38          rule19 = control.Rule(self.data_cpu['high'] &
    self.data_memory['low'] & self.data_disk['low'], self.data_status['low'])
39          rule20 = control.Rule(self.data_cpu['high'] &
    self.data_memory['low'] & self.data_disk['medium'],
40  self.data_status['medium'])
41          rule21 = control.Rule(self.data_cpu['high'] &
    self.data_memory['low'] & self.data_disk['high'], self.data_status['high'])
42          rule22 = control.Rule(self.data_cpu['high'] &
    self.data_memory['medium'] & self.data_disk['low'],
43  self.data_status['medium'])
44          rule23 = control.Rule(self.data_cpu['high'] &
    self.data_memory['medium'] & self.data_disk['medium'],
45  self.data_status['high'])
46          rule24 = control.Rule(self.data_cpu['high'] &
    self.data_memory['medium'] & self.data_disk['high'], self.status['very
47  high'])
48          rule25 = control.Rule(self.data_cpu['high'] &
    self.data_memory['high'] & self.data_disk['low'], self.status['very high'])
49          rule26 = control.Rule(self.data_cpu['high'] &
    self.data_memory['high'] & self.data_disk['medium'], self.status['very
50  high'])
51          rule27 = control.Rule(self.data_cpu['high'] &
    self.data_memory['high'] & self.data_disk['high'], self.status['very
52  high'])
53
```

Dalam Tabel 6.7 adalah proses aturan fungsi implikasi, ada 27 aturan berdasarkan Tabel 5.4. Aturan bisa menjadi penentu Controller dalam mengambil keputusan. Maka proses defuzzifikasi adalah seperti kode sumber berikut.

```
1  def compute(self, status_server):
2          self.fuzz_control.input['cpu'] = status_server['cpu']
3          self.fuzz_control.input['memory'] = status_server['memory']
4          self.fuzz_control.input['disk'] = status_server['disk']
5          self.fuzz_control.compute()
6          return self.fuzz_control.output['status']
7
8  def min(self, output):
9          return min(output, key=output.get)
```

Pada proses defuzzifikasi, setiap variabel input akan dihitung drajat keangotaannya dan menghasilkan keluaran berdasarkan pada baris ke 9 yaitu sumber daya terendah yang dihasilkan dari variabel input. Kemudian kode

sumber sebelumnya berlaku untuk *Controller* POX yang menggunakan bahasa pemrograman Python. Untuk dapat menjalankan aplikasi kita dapat menggunakan perintah berikut ini:

```
$ sudo python pox.py log.level --DEBUG Fuzzy --ip=10.0.0.100 --servers=10.0.0.11,10.0.0.12,10.0.0.13
```

Skrip Pox.py adalah Skrip utama yang ada pada controller Pox tersebut. Pox.py dijalankan agar dapat menjalan skrip fuzzy.py untuk mekanisme pembobotan server. Kemudian pada opsi log.level merupakan perintah untuk mendapatkan pesan yang ditampilkan dengan mode debug.

6.5 Implementasi Agen Psutils

Pada proses implementasi algoritme *Fuzzy* menggunakan *agen Psutil*, setiap server akan tanam program s*erver_agent.py* menggunakan library *psutil* untuk mendapatkan informasi status server dan *socket* sebagai media pengiriman data dengan protokol UDP.

6.5.1 *Instalasi* Psutils

Untuk dapat menjalankan agen Psutil, agar terlebih dahulu diinstall dengan perintah dibawah ini.

```
$ pip install psutil
```

6.5.2 Penerapan Algoritma Pada Agen Psutils

Penerapan Agen *Psutils* menggunakan socket UDP pada pemrograman Python. Bertujuan untuk mengirim informasi server yang diperoleh dari Psutil, untuk dikirim dalam bentuk CPU, memori dan disk dari setiap penggunaan sumber daya masing-masing server.

Tabel 6. 8 Kode Sumber Agen Psutils

```
1  import socket
2  import psutil
3  import pickle
4  import time
5  s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6  s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
7  server = {
9          'cpu': 0,
10         'memory': 0,
11         'disk': 0
12 }
13
14 def status_dari_server():
15         server ['cpu'] = psutil.cpu_percent(interval=1)
16         server ['memory'] = psutil.virtual_memory().percent
17         server ['disk'] = psutil.disk_usage('/').percent
```

```
18
19  if __name__ == '__main__':
20          print "Mengirimkan data ke controller"
21          while True:
22                  try:
23                          status_dari_server()
24                          s.connect(('10.0.0.100', 5000))
25                          s.send(pickle.dumps(server))
26                  except KeyboardInterrupt:
27                          break
28                  time.sleep(0.5)
```

Dalam Tabel 5.4 kita dapat menemukan kode sumber program Agen Psutils. Proses dimulai dengan memuat modul yang akan digunakan aplikasi yakni socket, psutil, pox dan waktu, seperti yang diperlihatkan pada baris 1 hingga baris 5. Selanjutnya, dibaris 6 dan 7 ada variabel untuk mempertahankan Alamat IP dan porta yang digunakan oleh protokol UDP. Berikut ini adalah inisialisasi soket UDP dengan opsi socket.SO_REUSEADDR, yang berguna untuk mengantisipasi penggunaan alamat IP yang sama. Oleh karena itu, program tersebut dapat menggunakan *internet protocol address* dan juga port yang sama apabila dihubungkan kembali ke pengontrol.

- Di baris 14 hingga 19 ada variabel di mana server memiliki status dari server.
- Dalam baris ke 22 hingga 25 terdapat fungsi get_server_status, yang berfungsi untuk memperoleh informasi tentang sumber daya server yang disisipkan di server melalui penggunaan modul psutil.
- Dalam baris 27 hingga 38 ada proses pengulangan yang mencakup beberapa proses, seperti mengambil informasi tentang status server, membungkus informasi menggunakan modul pickle dan mengirim data ke Controller.

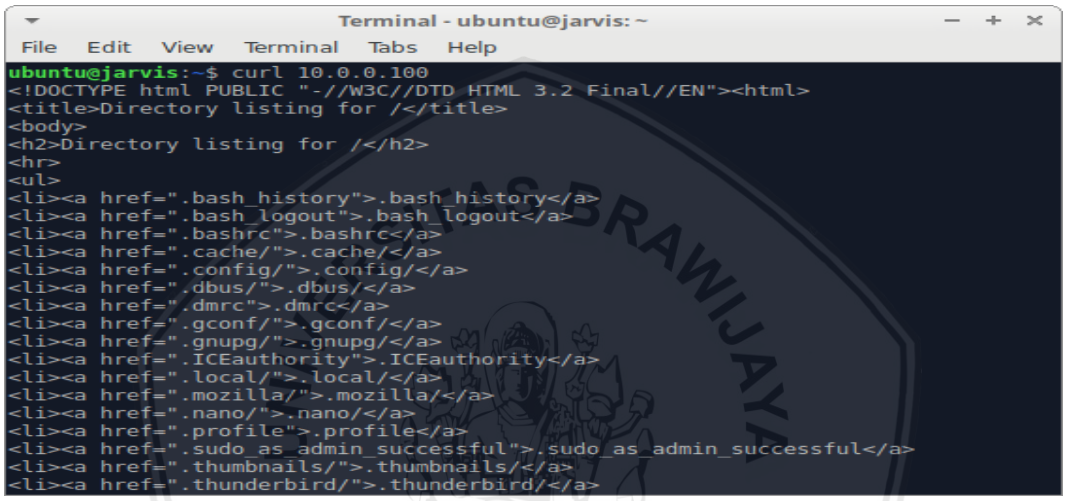
BAB 7 PENGUJIAN

7.1 Pengujian Fungsionalitas

7.1.1 Pengujian Fungsionalitas perangkat

Pada pengujian ini, perangkat yang digunakan pada sistem *load balancing* akan diuji agar mengetahui setiap perangkat dapat dijalankan. Adapun perangkat-perangkat yang digunakan yaitu:

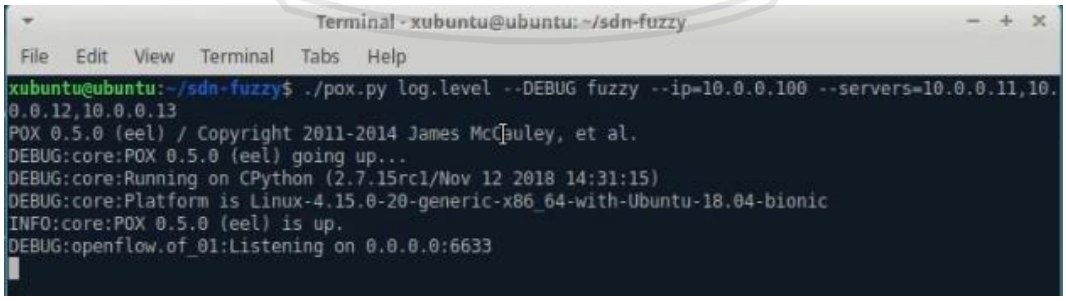
7.1.1.1 Klien



Gambar 7. 1 Tampilan klien

Perangkat ini digunakan untuk membantu proses pengujian pada penelitian ini, berdasarkan Gambar 7.1 yang menjelaskan pengiriman *request* terhadap server dengan *internet protocol address* 10.0.0.100 menggunakan *curl*. Kemudian ditanggapi oleh server berisikan admin_successful.

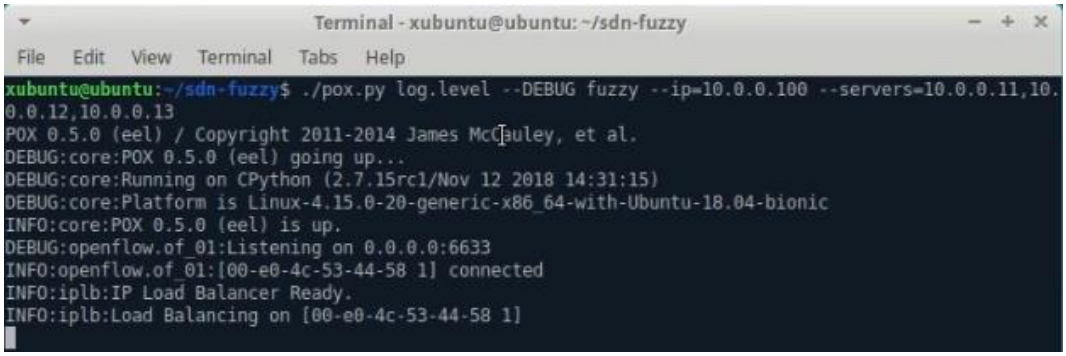
7.1.1.2 Controller



Gambar 7. 2 Tampilan Controller

Perangkat ini digunakan untuk melakukan proses pembobotan server menggunakan algoritme Fuzzy, serta pembagi beban trafik pada sistem *load balancing*. Berdasarkan Gambar 7.2 yang menjelaskan Controller dijalankan dalam keadaan up dan dapat terhubung ke dalam jaringan.

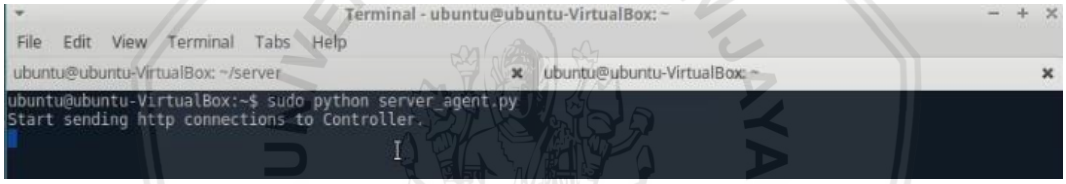
7.1.1.3 Switch



Gambar 7. 3 Tampilan Switch Terhubung

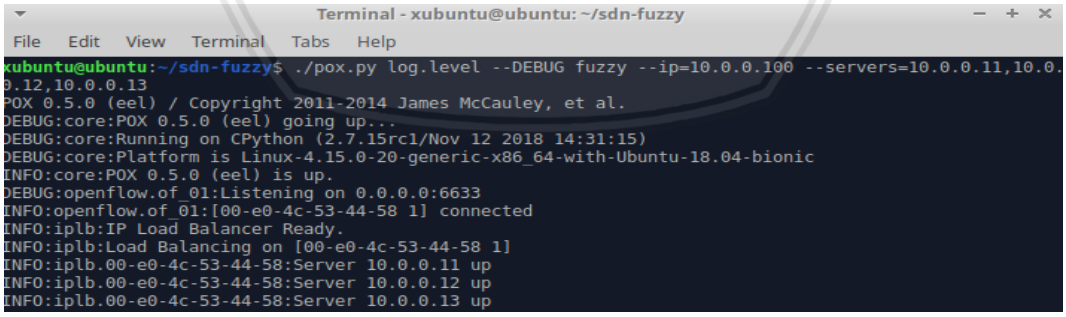
Perangkat ini digunakan sebagai penghubung antar komputer pada penelitian ini. Berdasarkan Gambar 7.3 yang menjelaskan Switch dapat dijalankan dan dapat terhubung ke Controller dengan pernyataan openflow terhubung (*connected*).

7.1.1.4 Server



Gambar 7. 4 Agen Mengirimkan Paket berupa Sumber Daya Server

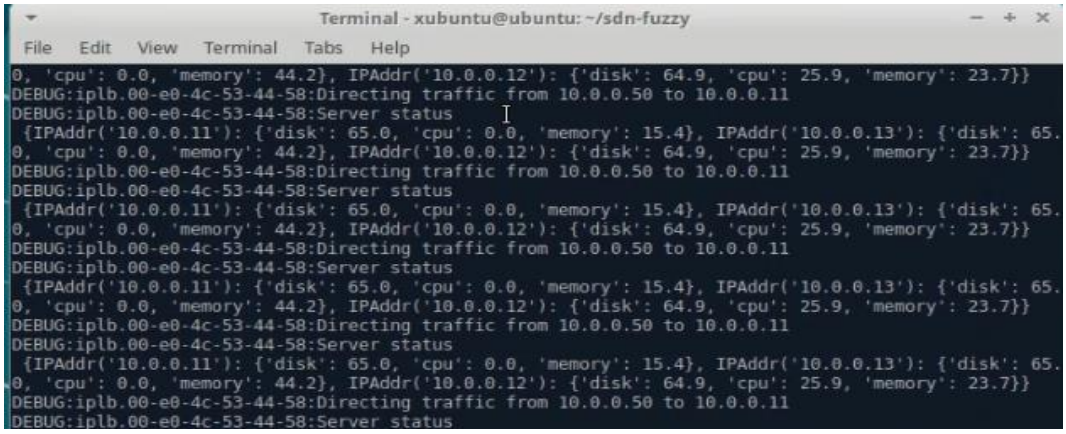
Perangkat ini digunakan sebagai penyedia layanan untuk klien, berdasarkan Gambar 7.4 menjelaskan tampilan dari server yang mengirimkan informasi berupa sumber daya server yang dimiliki oleh masing-masing server menggunakan agen Psutil.



Gambar 7. 5 Server Terhubung pada Controller

Kemudian, berdasarkan Gambar 7.5 menjelaskan tampilan dari contrroller yang menyatakan setiap server telah terhubung dengan ditandai dengan IP address dari masing-masing server up dan dapat melayani permintaan dari klien.

repository.ub.ac.id



```
0, 'cpu': 0.0, 'memory': 44.2}, IPAddr('10.0.0.12'): {'disk': 64.9, 'cpu': 25.9, 'memory': 23.7}}
DEBUG:iplb.00-e0-4c-53-44-58:Directing traffic from 10.0.0.50 to 10.0.0.11
DEBUG:iplb.00-e0-4c-53-44-58:Server status
 {IPAddr('10.0.0.11'): {'disk': 65.0, 'cpu': 0.0, 'memory': 15.4}, IPAddr('10.0.0.13'): {'disk': 65.
0, 'cpu': 0.0, 'memory': 44.2}, IPAddr('10.0.0.12'): {'disk': 64.9, 'cpu': 25.9, 'memory': 23.7}}
DEBUG:iplb.00-e0-4c-53-44-58:Directing traffic from 10.0.0.50 to 10.0.0.11
DEBUG:iplb.00-e0-4c-53-44-58:Server status
 {IPAddr('10.0.0.11'): {'disk': 65.0, 'cpu': 0.0, 'memory': 15.4}, IPAddr('10.0.0.13'): {'disk': 65.
0, 'cpu': 0.0, 'memory': 44.2}, IPAddr('10.0.0.12'): {'disk': 64.9, 'cpu': 25.9, 'memory': 23.7}}
DEBUG:iplb.00-e0-4c-53-44-58:Directing traffic from 10.0.0.50 to 10.0.0.11
DEBUG:iplb.00-e0-4c-53-44-58:Server status
 {IPAddr('10.0.0.11'): {'disk': 65.0, 'cpu': 0.0, 'memory': 15.4}, IPAddr('10.0.0.13'): {'disk': 65.
0, 'cpu': 0.0, 'memory': 44.2}, IPAddr('10.0.0.12'): {'disk': 64.9, 'cpu': 25.9, 'memory': 23.7}}
DEBUG:iplb.00-e0-4c-53-44-58:Directing traffic from 10.0.0.50 to 10.0.0.11
DEBUG:iplb.00-e0-4c-53-44-58:Server status
 {IPAddr('10.0.0.11'): {'disk': 65.0, 'cpu': 0.0, 'memory': 15.4}, IPAddr('10.0.0.13'): {'disk': 65.
0, 'cpu': 0.0, 'memory': 44.2}, IPAddr('10.0.0.12'): {'disk': 64.9, 'cpu': 25.9, 'memory': 23.7}}
DEBUG:iplb.00-e0-4c-53-44-58:Directing traffic from 10.0.0.50 to 10.0.0.11
DEBUG:iplb.00-e0-4c-53-44-58:Server status
```

Gambar 7. 6 Informasi Penggunaan Sumber daya Server

Setelah seluruh server mengirimkan informasi sumberdaya yang digunakan, berdasarkan Gambar 7.6 controller akan meneruskan permintaan dari klien yang diterima oleh server yang dinyatakan oleh IP address server dengan sumber daya terendah.

7.1.2 Pengujian Fungsionalitas Algoritme Fuzzy

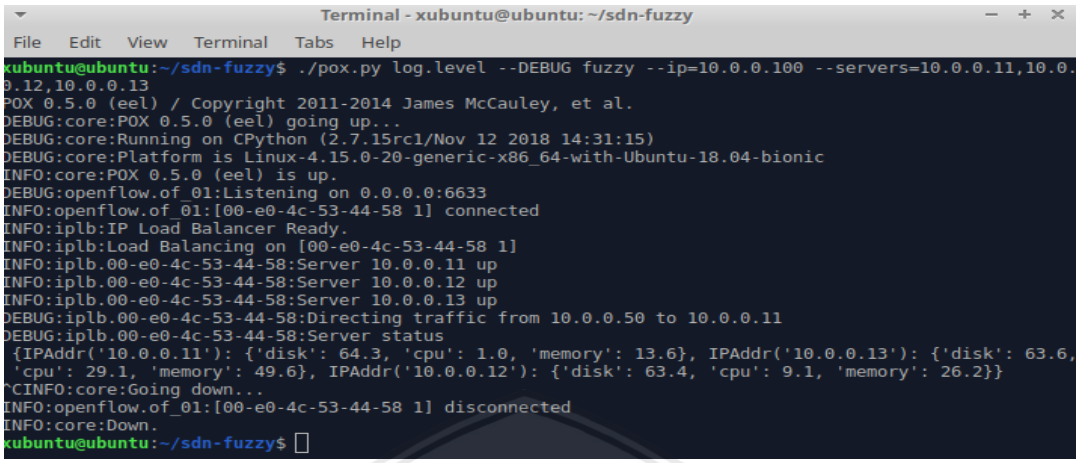
Pada proses pengujian ini, sistem *load balancing* diuji untuk mengetahui kinerja algoritme *Fuzzy* yanng ditambahkan metode agen Psutil. Pada prosesnya akan mengirimkan *request* ke tujuan http://10.0.0.100 sebagai alamat virtual *Web server.* Berdasarkan Gambar 7.1 yang menjelaskan pengiriman *request* terhadap server menggunakan *internet protocol address* 10.0.0.100 menggunakan *curl*. Sebelum pengiriman dilakukan, server telah melakukan pengiriman informasi berupa penggunaan sumber daya tiap server yang telah diterima controller.

Kemudian, mengacu pada Gambar 7.7 menampilkan informasi penggunaan sumber daya pada setiap server yang mana server 1 memiliki nilai CPU 1.0, *memory* 13.6 dan *disk 64.3,* server 2 memiliki nilai CPU 9.1, *memory* 26.2 dan *disk 63.4* dan server 3 memiliki nilai CPU 29.1, *memory* 49.6 dan *disk 63.3*, selanjutnya untuk dapat mengetahui bobot dari setiap server dilakukan mekanisme FIS mamdani sebagai berikut:

Langkah pertama melakukan proses inferensi berdasarkan aturan fuzzy seperti pada Tabel 5.4. Setelah itu, server 1 mengikuti aturan ke-2 yaitu, jika CPU low, memory low dan disk medium maka bobot server low. Kemudian server 2 mengikuti aturan ke-2 yaitu jika CPU low, memory low dan disk medium maka bobot server low. selanjutnya server 3 mengikuti aturan ke-4 yaitu jika CPU low, memory medium dan disk medium maka bobot server medium. Berdasarkan aturan fuzzy, server 3 memiliki bobot medium yang artinya Controller tidak meneruskan request terhadap server 3. Sedangkan server 1 dan 2 memiliki bobot yang sama yaitu low. Oleh sebab itu, dibutuhkan perhitungan fuzzy menggunakan metode FIS mamdani untuk mendapatkan hasil bobot server yang paling rendah. Setelah mendapatkan nilai bobot server, maka Controller akan

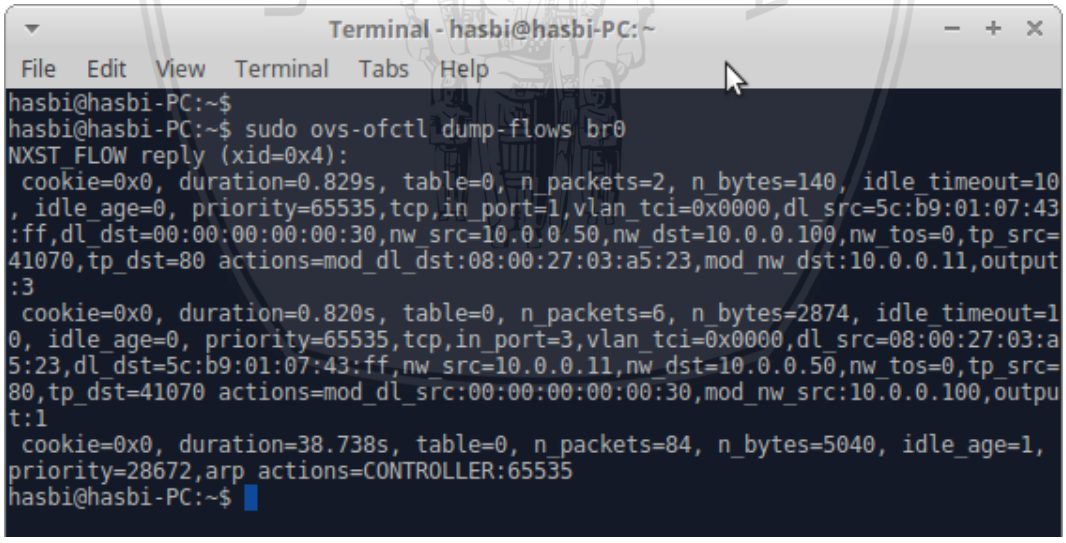
UNIVERSITAS BRAWIJAYA

meneruskan request pada server yang memiliki bobot server terendah seperti pada Gambar 7.7 dibawah ini.



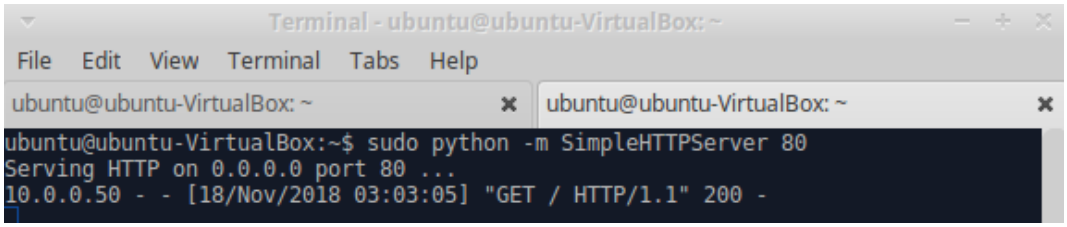
Gambar 7. 7 Controller Meneruskan Traffic pada Server 1

Gambar 7.7 menjelaskan hasil dari kinerja *Controller* saat menerima request dari client. Dari hasil tersebut, *controller* meneruskan *request* ke server 1. Hal ini disebabkan, server 1 memiliki penggunaan sumber daya paling rendah dibandingkan dengan server lainnya dengan nilai CPU 1.0, *memory* 13.6 dan *disk 64.3*, dari nilai penggunaan inilah proses fuzzifikasi dan defuzzifikasi berlangsung. Selain itu juga, controller mengirimkan data berupa kondisi server lainnya.

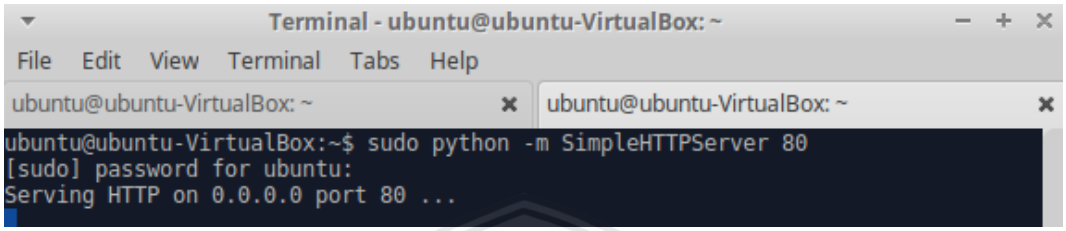


Gambar 7. 8 Table Openvswitch

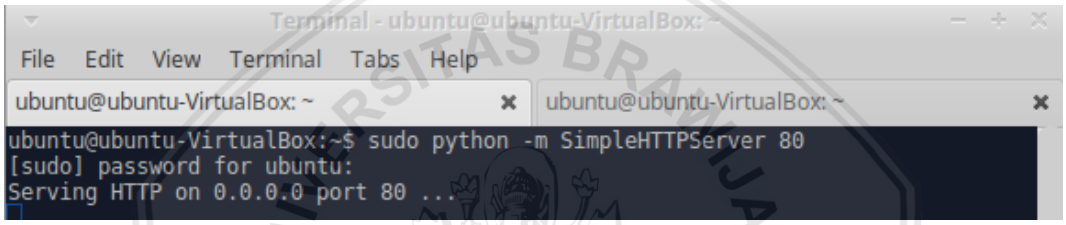
Gambar 7.8 adalah terminal Switch berupa aliran atau *Flow* dari *flow table.* Data tersebut membuktikan bahwa algoritme *Fuzzy* dapat menentukan traffik yang sudah ditentukan oleh *controller* dan meneruskan paket pada penggunaan sumber daya paling rendah.



Gambar 7. 9 Terminal *Server* dengan Sumber daya Tinggi



Gambar 7. 10 Terminal Server dengan Sumber daya Sedang



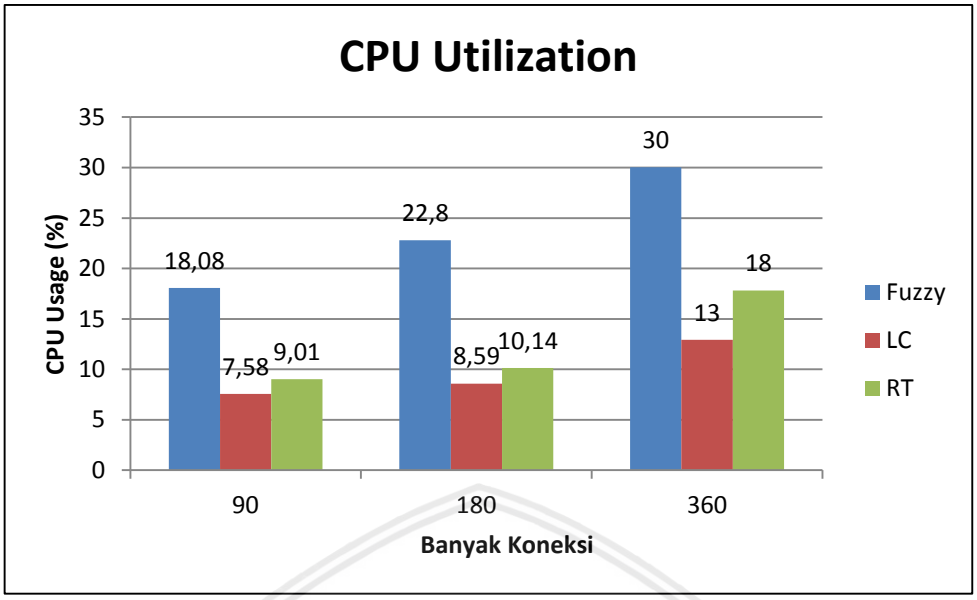
Gambar 7. 11 Terminal *Server* dengan Sumber daya Rendah

Kemudian paket tersebut akan diterima oleh server 1 yang memiliki penggunaan sumber daya paling rendah dengan IP Adress 10.0.0.11 yang berada di Gambar 7.9. Adapun pada Gambar 7.10 dan 7.11 merupakan server 2 dan 3, yang mana kedua server tersebut tidak menerima request dari *controller*.

7.1.3 Pengujian Kinerja Controller

Pada pengujian ini, perangkat *Controller* yang digunakan pada sistem *load balancing* menggunakan algoritme Fuzzy akan diuji agar mengetahui kinerja dari *Controller* tersebut.

Pada Gambar 7.12 merupakan grafik perbandingan pengujian CPU *usage* di *Controller.* Algoritme LC memiliki hasil dari CPU *usage* paling rendah pada setiap skenario. Hal ini disebabkan, algoritme LC tidak melalui mekanisme perhitungan yang dapat mengurangi beban kinerja dari *Controller*. Kemudian CPU *usage* paling tinggi dimiliki oleh algoritme Fuzzy. Hal ini disebabkan, algoritme fuzzy melakukan perhitungan logika fuzzy pada setiap data server berupa informasi penggunaan sumber daya server yang didapatkan dari masing-masing server untuk menentukan server yang akan menerima request. Oleh karena itu, beban kinerja dari Controller yang menggunakan algoritme Fuzzy akan tinggi dibandingkan dengan algoritme LC dan RT.



Gambar 7. 12 Grafik CPU *Usage Controller*

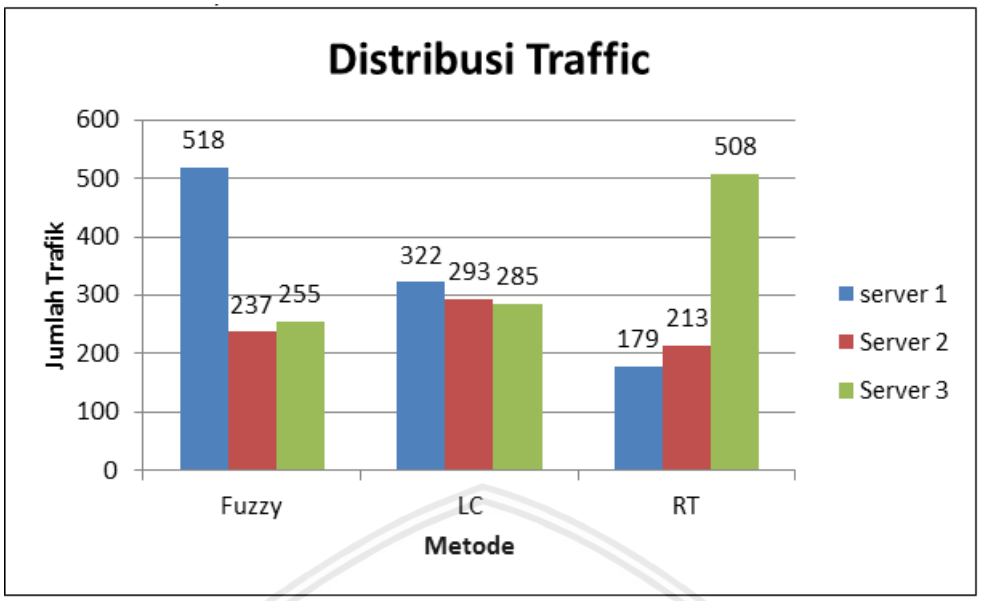
7.2 Pengujian Kinerja Tanpa Beban

Pada Pengujian kinerja tanpa beban, tiap server dalam keadaan low dan bertujuan untuk mendapatkan hasil dari kinerja *load balancing* menggunakan perangkat lunak HTTPerf berdasarkan tiga skenario. Adapun skenario dari pengujian yaitu: rate 90 (*low)*, rate 180 (*medium)* dan rate 360 (*high)*. Kemudian HTTPerf akan mengirimkan request secara langsung ke *web server* dengan alamat http://10.0.0.100.

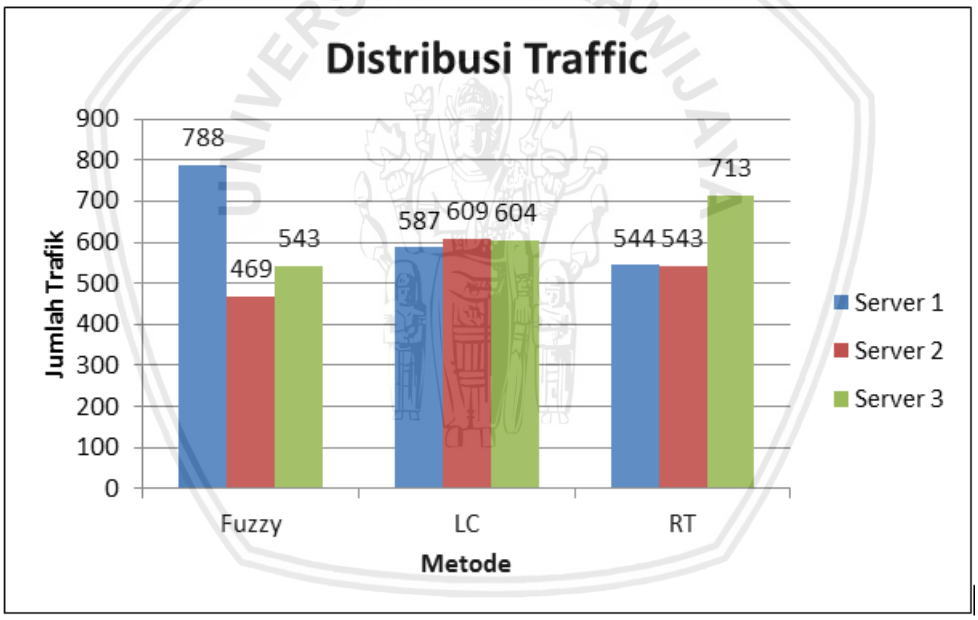
7.2.1 Pengujian Distribusi Traffic

Pada setiap metode yang digunakan memiliki mekanisme untuk menndistribusikan beban berdasarkan kemampuan dari metode itu sendiri. Adapun tujuan dari pengujian ini untuk membuktikan bahwa algoritme *Fuzzy* yang diterapkan pada arsitektur SDN mampu mendistribusikan *traffic* ke beberapa server. Kemudian akan dibandingkan dengan algoritme LC dan RT.

Dalam Gambar 7.13 menjelaskan bahwa kategori low pendistribusian traffic yang paling seimbang dimiliki oleh algoritme *Least connection,* hal ini disebabkan algoritme LC membagi traffic berdasarkan koneksi terkecil. Kemudian algoritme *Fuzzy* membagi beban berdasarkan penggunaan sumber daya paling rendah yang akan membebani server 1 yang memiliki sumber daya server paling besar dibandingkan server lainnya. Selanjutnya pada algoritme *Response time* (RT) membagi trafik berdasarkan *response time* paling tinggi. Oleh sebab itu, server 3 memiliki beban paling besar dibandingkan server lainnya.

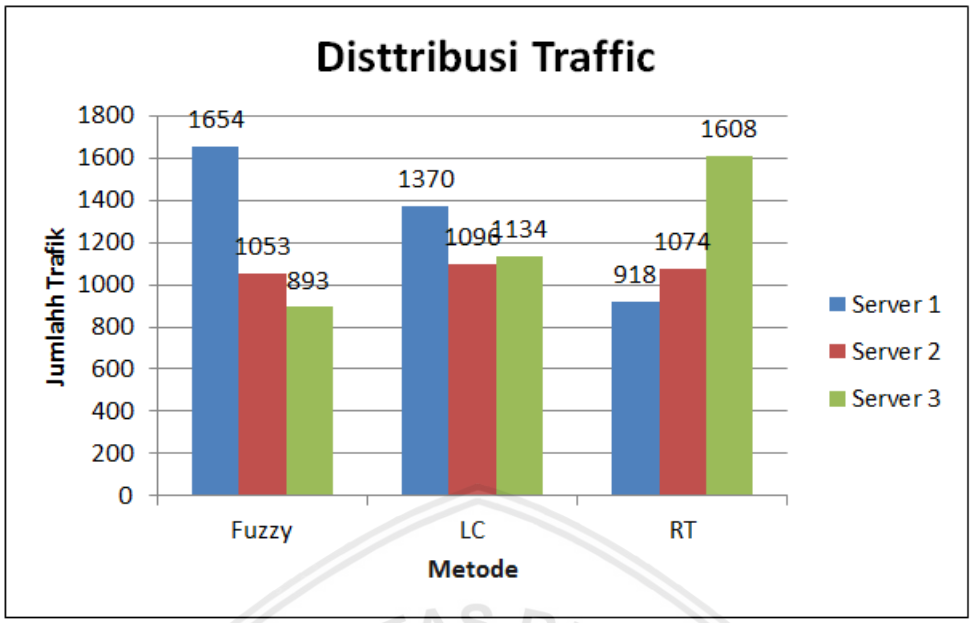


Gambar 7. 13 Grafik Distribusi Trafik Pengujian Tanpa Beban Skenario Low



Gambar 7. 14 Grafik Distribusi Trafik Pengujian Tanpa Beban Skenario Medium

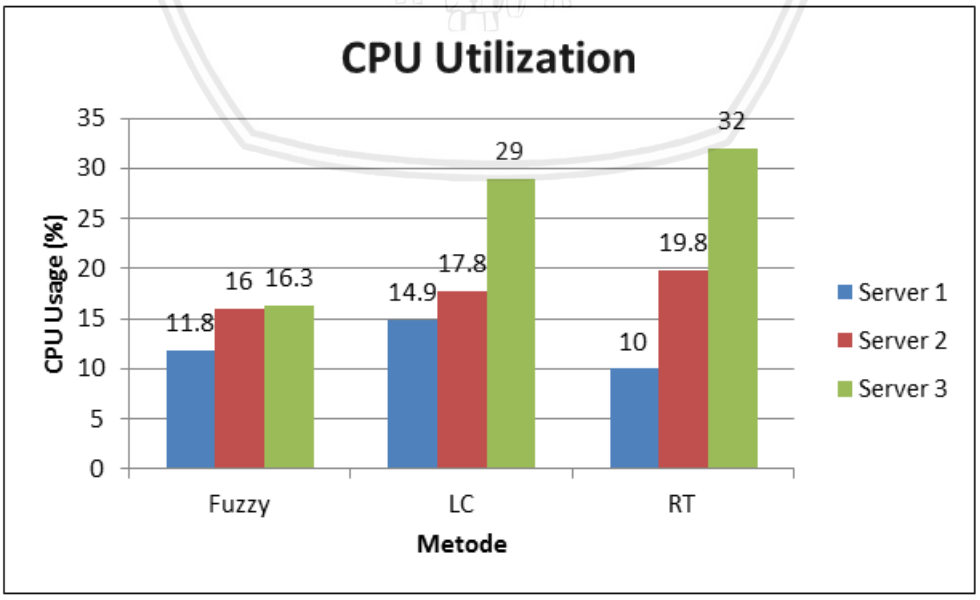
Gambar 7.14 menjelaskan bahwa kategori medium pendistribusian traffic yang paling seimbang dimiliki oleh algoritme *Least connection,* hal ini disebabkan algoritme LC membagi traffic berdasarkan koneksi terkecil. Kemudian algoritme *Fuzzy* membagi beban berdasarkan penggunaan sumber daya paling rendah yang akan membebani server 1 yang memiliki sumber daya server paling besar dibandingkan server lainnya. Selanjutnya pada algoritme *Response time* (RT) membagi trafik berdasarkan *response time* paling tinggi. Oleh sebab itu, server 3 memiliki beban paling besar dibandingkan server lainnya.



Gambar 7. 15 Grafik Distribusi Trafik Pengujian Tanpa Beban Skenario High

Gambar 7.15 menjelaskan bahwa kategori high pendistribusian traffic yang paling seimbang dimiliki oleh algoritme *Least connection,* hal ini disebabkan algoritme LC membagi traffic berdasarkan koneksi terkecil. Kemudian algoritme *Fuzzy* membagi beban berdasarkan penggunaan sumber daya paling rendah yang akan membebani server 1 yang memiliki sumber daya server paling besar dibandingkan server lainnya. Selanjutnya pada algoritme *Response time* (RT) membagi trafik berdasarkan *response time* paling tinggi. Oleh sebab itu, server 3 memiliki beban paling besar dibandingkan server lainnya.

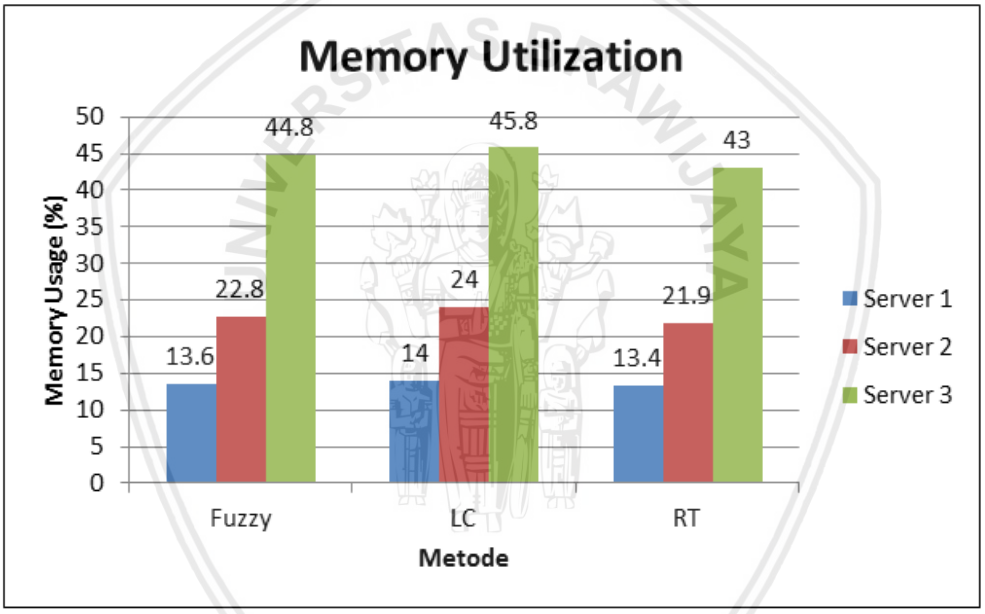
7.2.2 Pengujian Resource (CPU dan Memory)



Gambar 7. 16 Grafik CPU Usage pengujian Tanpa Beban Skenario Low

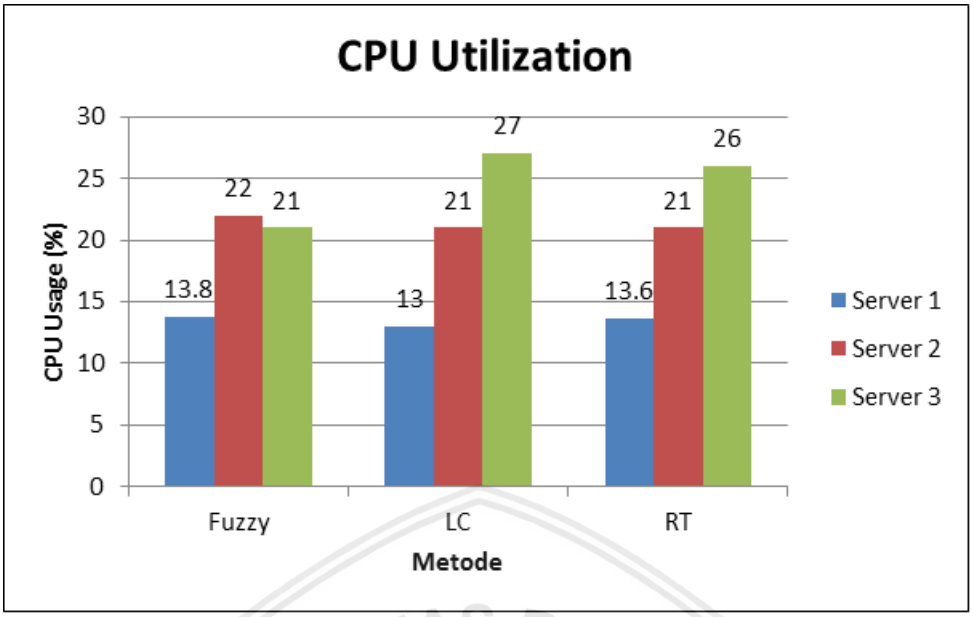
Pada Gambar 7.16 merupakan grafik perbandingan pengujian tanpa beban CPU *usage* di kategori *low.* Algoritme response time (RT) memiliki hasil dari CPU usage paling rendah pada server 1 dengan nilai 10%. Hal ini disebabkan, server 1 pada algoritme RT mendapatkan pendistibusian trafik paling sedikit dibandingkan server 1 pada algoritme *Fuzzy* dan LC. Kemudian CPU *usage* paling rendah diserver 2 yang memiliki nilai 16 % dan pada server 3 yang memiliki 16,3 % dimiliki oleh algoritme Fuzzy. Hal ini disebabkan, algoritme fuzzy mendapatkan pendistibusian trafik pada server 2 dan 3 lebih sedikit dibandingkan algoritme LC dan RT.

Pada Gambar 7.17 merupakan hasil dari pengujian tanpa beban *memory usage* di kategori *low.* Algoritme *Response time* memiliki hasil memory usage yang paling rendah diseluruh server dibandingkan dengan algoritme *Fuzzy* dan LC. Terjadinya hal ini, karena algoritme RT tidak menerapkan agen *Psutil* di setiap server yang menyebabkan menurunkan penggunaan *memory*.

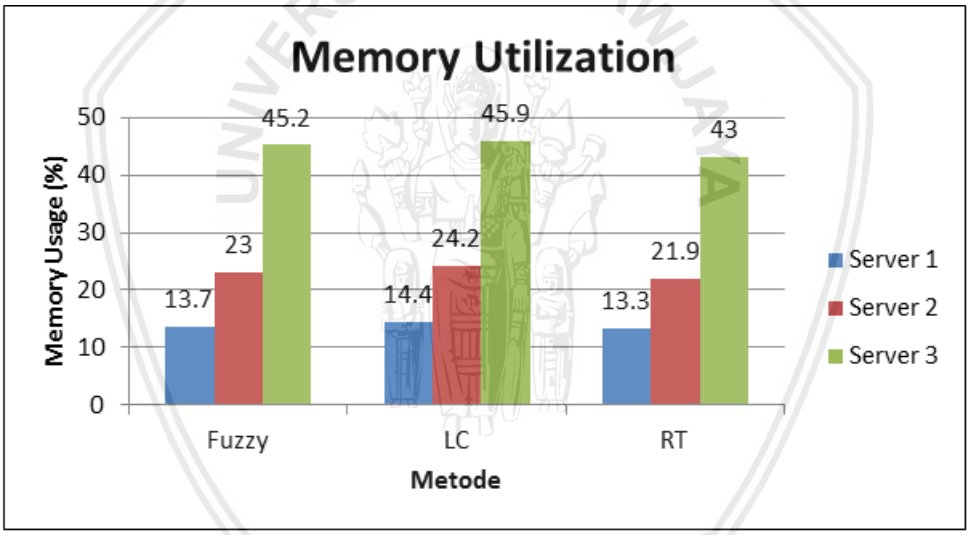


Gambar 7. 17 Grafik Memory Usage pengujian Tanpa Beban Skenario Low

Pada Gambar 7.18 merupakan grafik perbandingan pengujian tanpa beban CPU *usage* di kategori *medium.* Algoritme *Least connection* (LC) memiliki hasil dari CPU usage paling rendah pada server 1 dengan nilai 13%. Kemudian CPU *usage* paling rendah di server 2 dengan nilai 21 % dimiliki oleh algoritme RT. Selanjutnya CPU *usage* paling rendah di server 3 dengan nilai 21 % dimiliki oleh algoritme *Fuzzy*. Hal ini disebabkan, algoritme fuzzy mendapatkan pendistibusian trafik pada server 3 lebih sedikit dibandingkan algoritme LC dan RT.

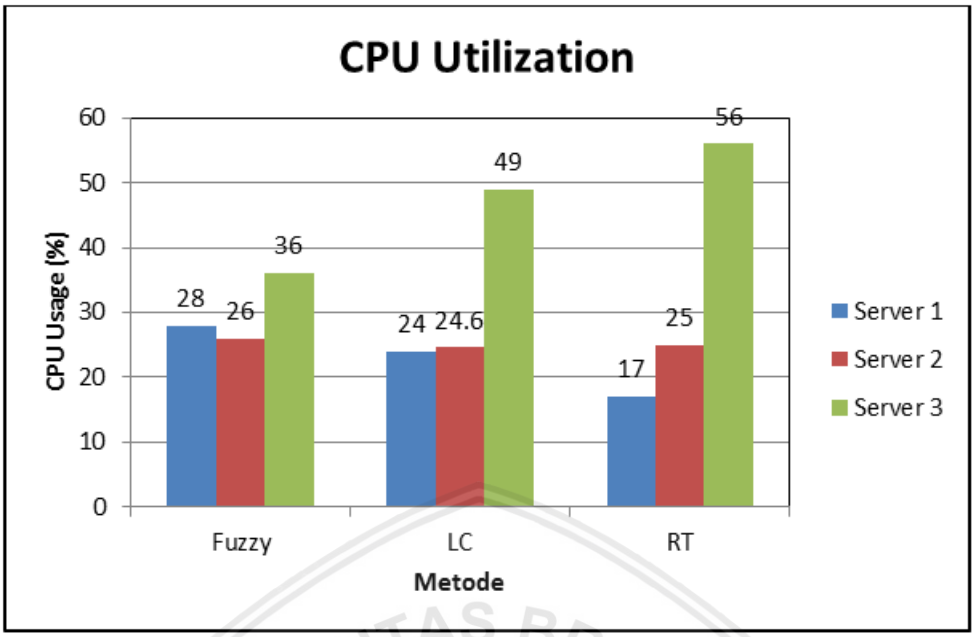


Gambar 7. 18 Grafik CPU Usage pengujian Tanpa Beban Skenario Medium



Gambar 7. 19 Grafik Memory Usage pengujian Tanpa Beban Skenario Medium

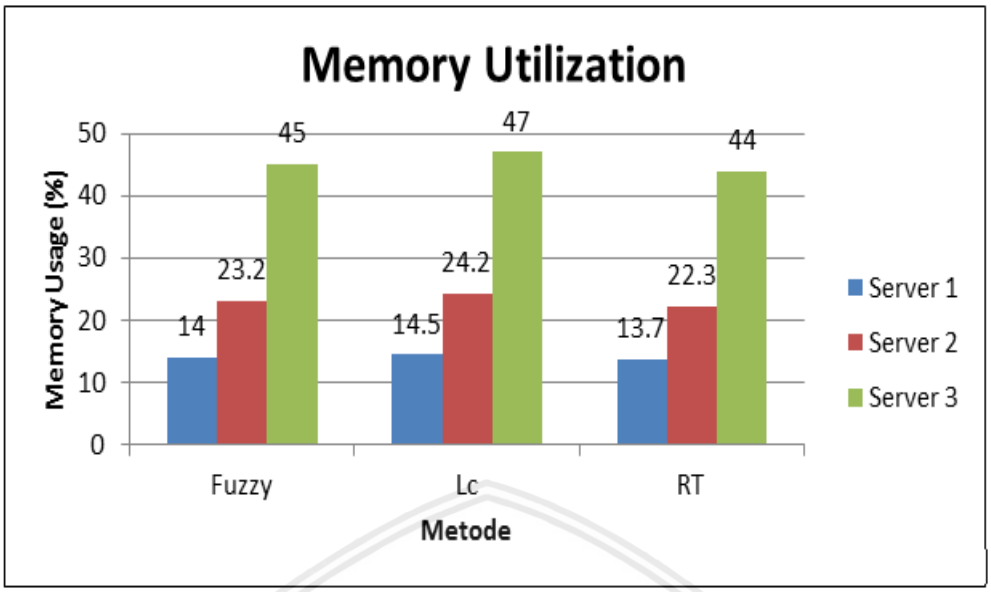
Pada Gambar 7.19 merupakan hasil dari pengujian tanpa beban *memory usage* di kategori *medium.* Algoritme *Response time* memiliki hasil memory usage yang paling rendah diseluruh server dibandingkan dengan algoritme *Fuzzy* dan LC. . Terjadinya hal ini, karena algoritme RT tidak menerapkan agen *Psutil* di setiap server yang menyebabkan menurunkan penggunaan *memory*.



Gambar 7. 20 Grafik CPU Usage pengujian Tanpa Beban Skenario High

Pada Gambar 7.20 merupakan grafik perbandingan pengujian tanpa beban CPU *usage* di kategori *high.* Algoritme response time (RT) memiliki hasil dari CPU usage paling rendah pada server 1 dengan nilai 17%. Hal ini disebabkan, server 1 pada algoritme RT mendapatkan pendistibusian trafik paling sedikit dibandingkan server 1 pada algoritme *Fuzzy* dan LC. Kemudian penggunaan CPU paling rendah pada server 2 dengan nilai 24,6 % dimiliki oleh algoritme LC. Selanjutnya penggunaan CPU paling rendah pada server 3 dengan nilai 28 % dimiliki oleh algoritme *Fuzzy*. Hal ini disebabkan, server 3 pada kategori high mendapatkan pendistibusian trafik paling sedikit dibandingkan server 1 pada algoritme LC dan RT.

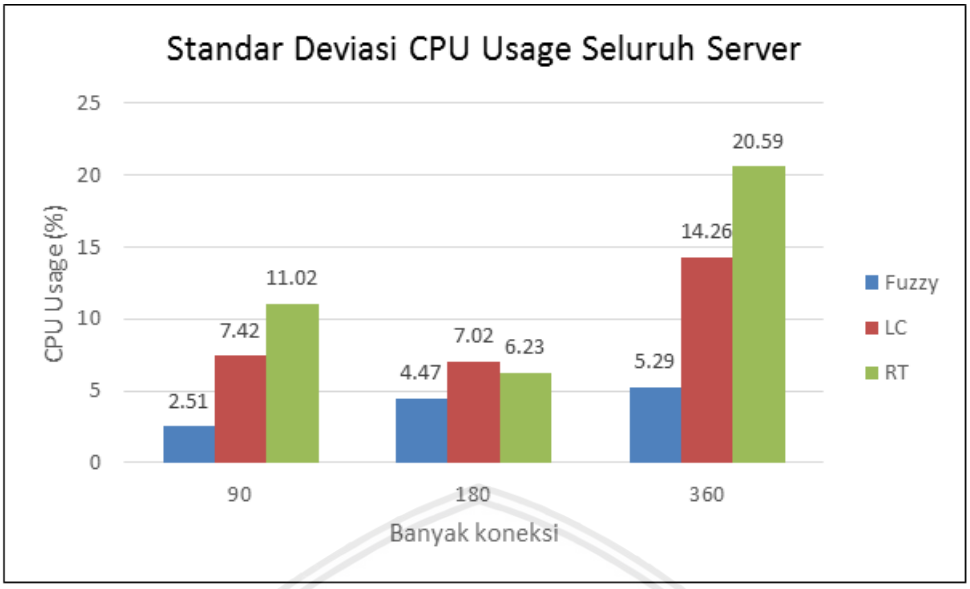
Pada Gambar 7.21 merupakan hasil dari pengujian tanpa beban *memory usage* di kategori *high.* Algoritme *Response time* memiliki hasil memory usage yang paling rendah diseluruh server dibandingkan dengan algoritme *Fuzzy* dan LC. . Terjadinya hal ini, karena algoritme RT tidak menerapkan agen *Psutil* di setiap server yang menyebabkan menurunkan penggunaan *memory.*



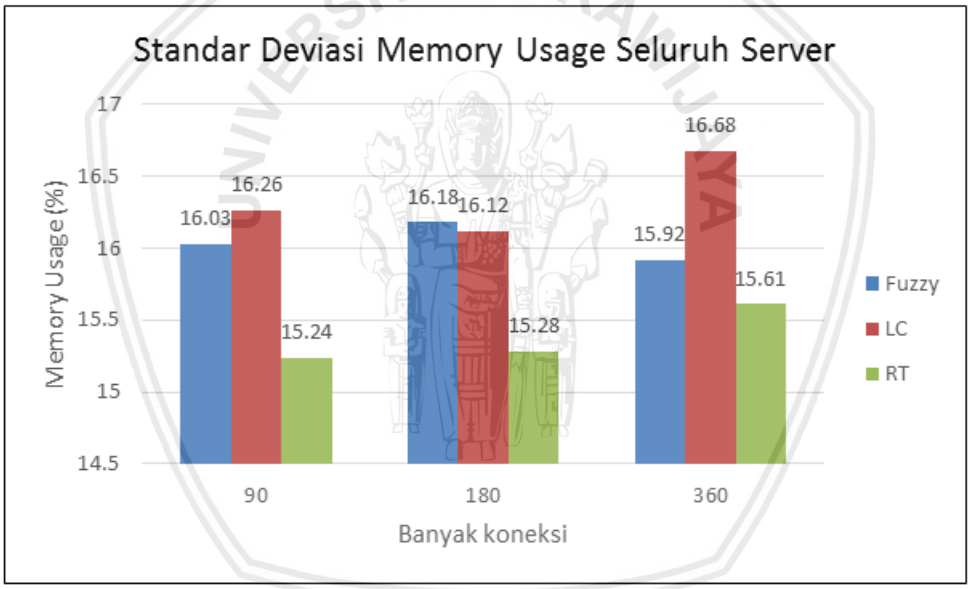
Gambar 7. 21 Grafik Memory Usage pengujian Tanpa Beban Skenario High

Pada Gambar 7.22 menunjukan grafik standar deviasi dari CPU *Usage*. Metode fuzzy memperoleh nilai paling rendah serta menegaskan dapat mendistribusikan trafik secara seimbang berdasarkan penggunaan sumber daya server terkecil. CPU merupakan salah satu variabel dalam penentuan pendistribusian trafik menggunakan metode *Fuzzy.* Kemudian metode *Least Connection* memiliki standart deviasi lebih tinggi dibandingkan dengan metode *Fuzzy.* Hal ini dipengaruhi oleh pendistribusian trafik metode LC berdasarkan koneksi terkecil yang mana setiap server mendapatkan trafik yang seimbang. Selanjutnya metode *Response Time* memiliki standart deviasi yang paling tinggi. Hal ini dipengaruhi oleh pendistribusian trafik yang didasari *response* server yang paling cepat tanpa memperdulikan sumber daya server. Oleh karena itu, meyebabkan kondisi server yang memiliki sumber daya yang rendah dapat terbebani dibandingkan dengan jumlah sumber daya yang tinggi.

Pada Gambar 7.23 merupakan standar deviasi *Memory Usage.* Metode *response time* memiliki nilai yang paling rendah. Hal ini dikarenakan metode response time tidak menggunakan *Agen Psutil* yang mana mengurangi penggunaan sumber daya memory server. Kemudia metode *Fuzzy* memiliki nilai yang lebih rendah dari metode *Least Connection.* Hal ini menandakan bahwa metode tersebut mendistribusikan trafik secara adil berdasarkan sumber daya memory. *Memory* merupakan salah satu variabel dalam penentuan pendistribusian trafik menggunakan metode *Fuzzy.* Selanjutnya algoritma LC memiliki nilai standar deviasi *memory usage* yang paling buruk.



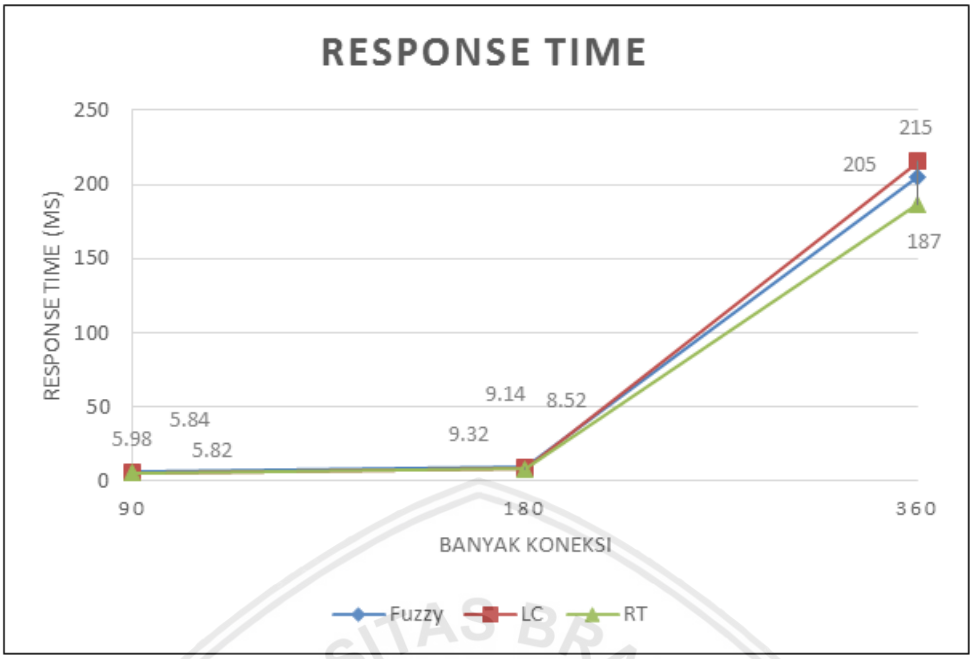
Gambar 7. 22 Grafik Standar Devisiasi CPU Usage Pengujian Tanpa Beban



Gambar 7. 23 Grafik Standar Devisiasi CPU Usage Pengujian Tanpa Beban

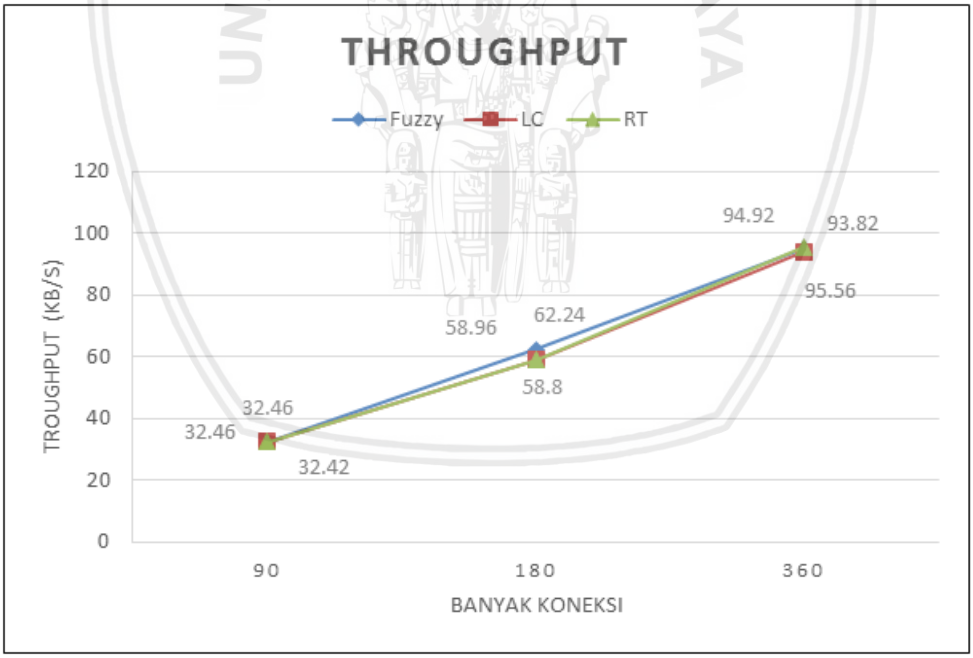
7.2.3 Pengujian Response Time

Pengujian dengan parameter *Response time* pada pengujian tanpa beban yang bertujuan mendapatkan hasil *Response time* pada metode *Fuzzy* yang dibandingkan dengan algoritme LC dan RT. Pada setiap algoritme *Fuzzy*, LC, dan RT mendapatkan hasil yang berbeda pada tiap skenario pengujian *90, 180* dan 360. Hal ini disebabkan semakin banyaknya *request* yang dilayani oleh masing-masing server, maka semakin banyak koneksi yang terjalin. Berdasarkan Gambar 7.24 algoritme RT lebih baik dibandingkan algoritme *Fuzzy* dan LC pada kategori *low* dengan nilai 5,82 ms, kategori *medium* dengan nilai 8,54 ms dan kategori *high* dengan nilai 187 ms. Hal ini disebabkan metode response time membagi traffic berdasarkan *response time* tercepat dari setiap server.



Gambar 7. 24 Grafik Response Time pengujian Tanpa Beban

7.2.4 Pengujian Troughput



Gambar 7. 25 Grafik Troughput pengujian Tanpa Beban

Pengujian *Troughput* pada pengujian tanpa beban yang bertujuan untuk mengetahui hasil *Troughput* dari algoritme *Fuzzy* untuk dibandingkan pada algoritme LC dan RT. Pada setiap algoritme *Fuzzy*, LC, dan RT mendapatkan hasil yang berbeda di setiap skenario pengujian 90, 180 dan 360. Hal ini disebabkan semakin banyaknya *request* yang dilayani server, maka semakin tinggi hasil dari *Troughput* itu sendiri. Berdasarkan Gambar 7.25 menjelaskan kategori *low*

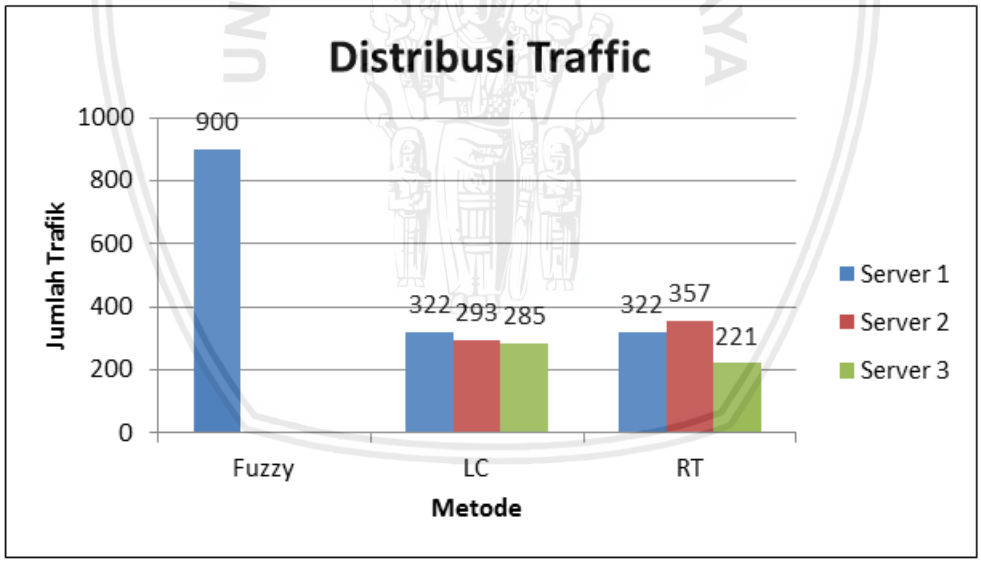
dengan nilai 32.42 KB/s, kategori *medium* dengan nilai 58,8 KB/s dan kategori *high* dengan nilai 93,82 dihasilkan oleh algoritme RT lebih tinggi dibandingkan algoritme *Fuzzy* dan LC. Hal ini dikarenakan semakin kecilnya *response time* maka semakin besar *troughput* yang dihasilkan.

7.3 Pengujian Kinerja Dengan Beban

Dalam Pengujian kinerja, masing-masing server dibebankan dengan pemutaran 2 video. Oleh sebab itu, server 1 dalam keadaan *low*, server 2 dalam keadan *medium* dan server 3 dalam *high* yang bertujuan untuk mendapatkan hasil dari kinerja *load balancing* menggunakan perangkat lunak HTTPerf berdasarkan tiga skenario. Adapun skenario dari pengujian yaitu: rate 90 (*low)*, rate 180 (*medium)* dan rate 360 (*high)*. Kemudian HTTPerf akan mengirimkan request secara langsung ke *web server* dengan alamat http://10.0.0.100.

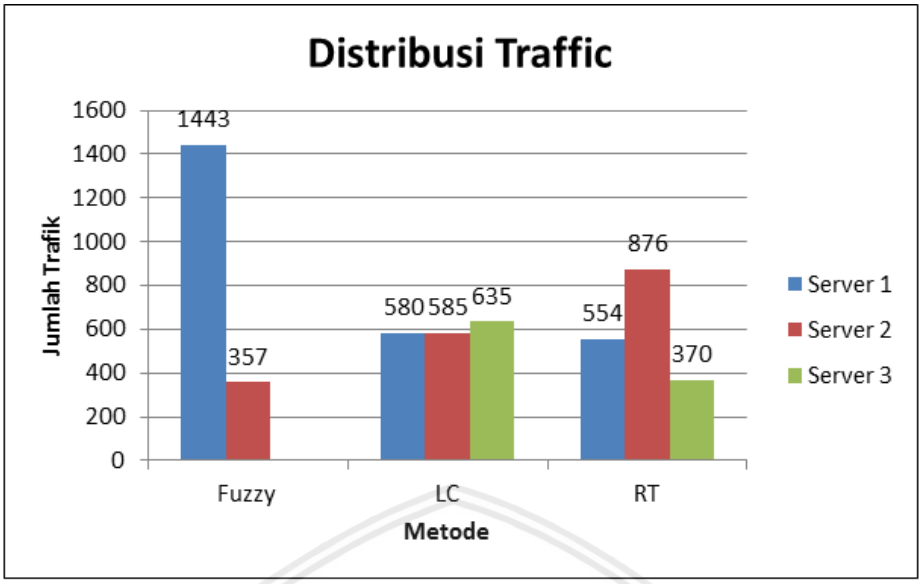
7.3.1 Pengujian Distribusi Trafik

Pada setiap metode yang digunakan memiliki mekanisme untuk menndistribusikan beban berdasarkan kemampuan dari metode itu sendiri. Adapun tujuan dari pengujian ini untuk membuktikan bahwa algoritme *Fuzzy* yang diterapkan pada arsitektur SDN mampu mendistribusikan *traffic* ke beberapa server. Kemudian akan dibandingkan dengan algoritme LC dan RT.



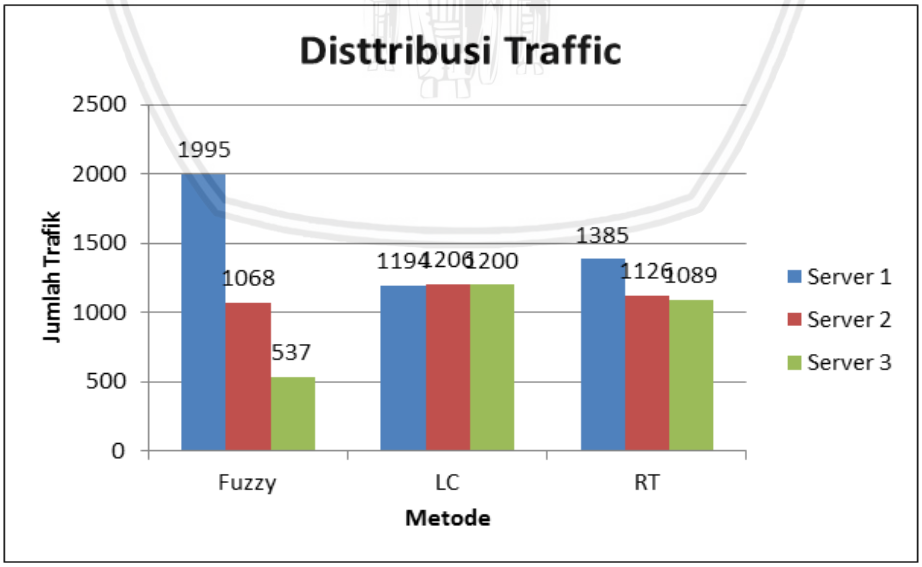
Gambar 7. 26 Grafik Distribusi Trafik Pengujian Dengan Beban Skenario Low

Gambar 7.26 menjelaskan bahwa kategori low pendistribusian traffic yang paling seimbang dimiliki oleh algoritme *Least connection,* hal ini disebabkan algoritme LC membagi trafik berdasarkan koneksi terkecil. Kemudian algoritme *Fuzzy* membagi beban berdasarkan penggunaan sumber daya paling rendah yang akan membebani server 1 yang memiliki sumber daya server paling besar dibandingkan server lainnya. Selanjutnya pada algoritme *Response time* (RT) membagi trafik berdasarkan *response time* paling tinggi. Oleh sebab itu, server 2 memiliki beban paling besar dibandingkan server lainnya.



Gambar 7. 27 Grafik Distribusi Trafik Pengujian Dengan Beban Skenario Medium

Gambar 7.27 menjelaskan bahwa kategori *medium* pendistribusian traffic yang paling seimbang dimiliki oleh algoritme *Least connection* (LC), hal ini disebabkan algoritme LC membagi trafik berdasarkan koneksi terkecil. Kemudian algoritme *Fuzzy* membagi beban berdasarkan penggunaan sumber daya paling rendah yang akan membebani server 1 yang memiliki sumber daya server paling besar dibandingkan server lainnya. Selanjutnya pada algoritme *Response time* (RT) membagi trafik berdasarkan *response time* paling tinggi. Oleh sebab itu, server 2 memiliki beban paling besar dibandingkan server lainnya.

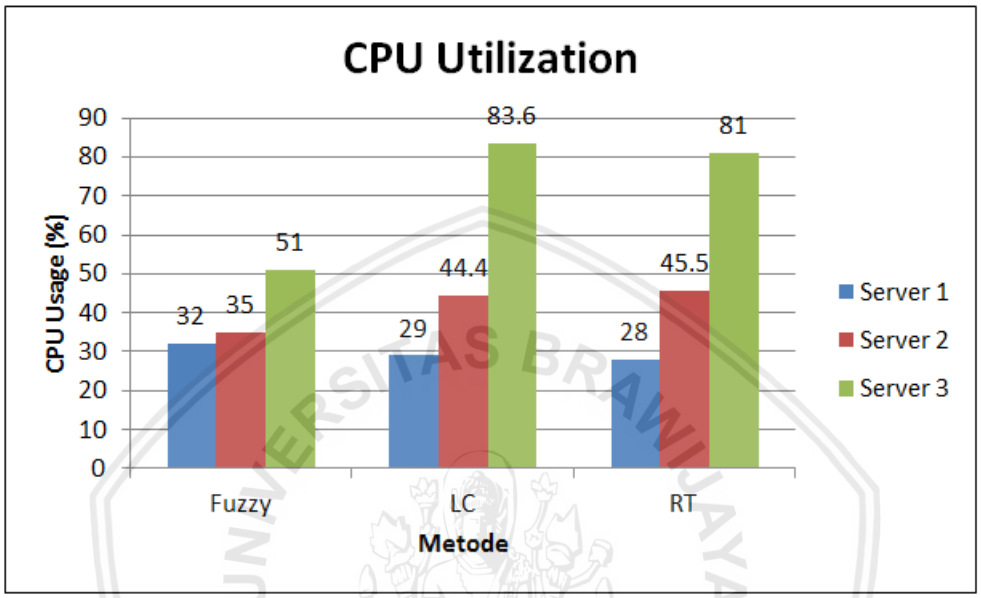


Gambar 7. 28 Grafik Distribusi Trafik Pengujian Dengan Beban Skenario High

Gambar 7.28 menjelaskan bahwa kategori *high* pendistribusian trafik yang paling seimbang dimiliki oleh algoritme *Least connection,* hal ini disebabkan

algoritme LC membagi trafik berdasarkan koneksi terkecil. Kemudian algoritme *Fuzzy* membagi beban berdasarkan penggunaan sumber daya paling rendah yang akan membebani server 1 yang memiliki sumber daya server paling besar dibandingkan server lainnya. Selanjutnya pada algoritme *Response time* (RT) membagi trafik berdasarkan *response time* paling tinggi.

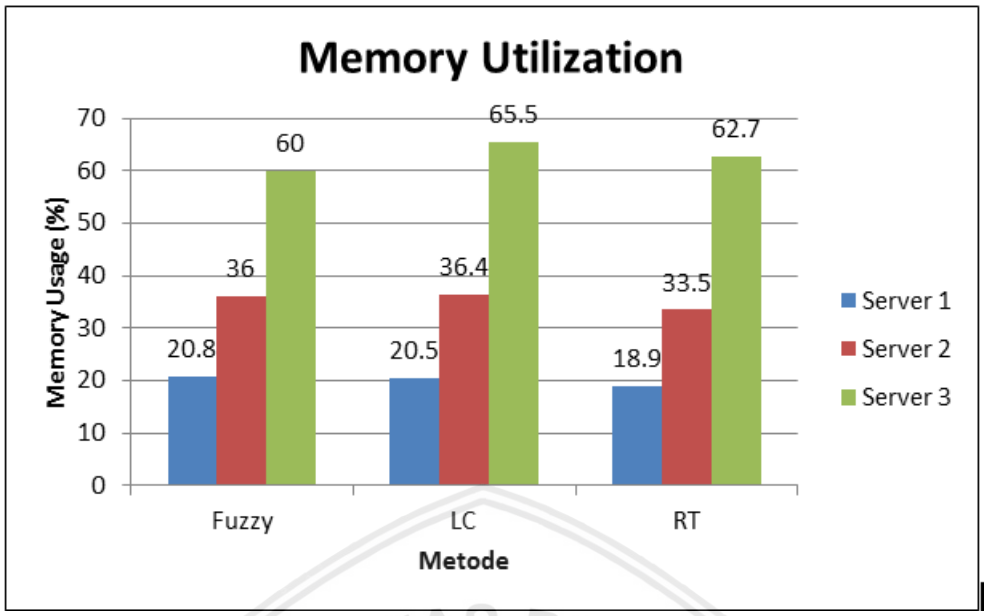
7.3.2 Pengujian Resource (CPU dan Memory)



Gambar 7. 29 Grafik CPU Usage Pengujian Dengan Beban Skenario Low

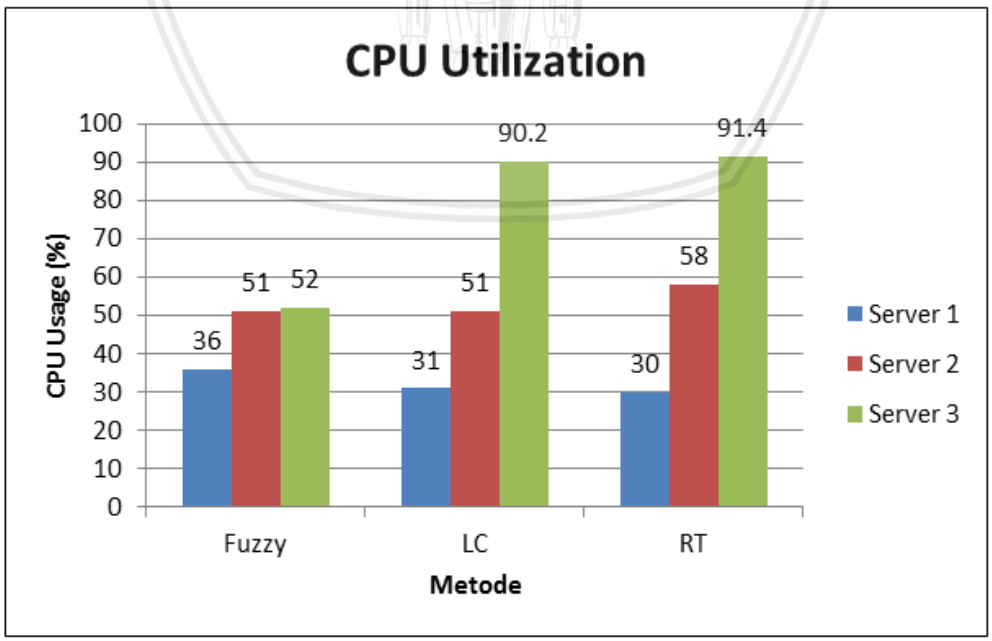
Pada Gambar 7.29 merupakan grafik perbandingan pengujian kinerja dengan beban CPU *usage* di kategori *low.* Algoritme *response time* (RT) memiliki hasil dari CPU usage paling rendah pada server 1 dengan nilai 28%. Hal ini disebabkan, server 1 pada algoritme RT mendapatkan pendistibusian trafik paling sedikit dibandingkan server 1 pada algoritme *Fuzzy* dan LC. Kemudian CPU *usage* paling rendah diperoleh server 2 yang memiliki nilai 35 % dan pada server 3 yang memiliki nilai 44,4 % dimiliki oleh algoritme Fuzzy. Hal ini disebabkan, algoritme *Fuzzy* mendapatkan pendistibusian trafik pada server 2 dan 3 lebih sedikit dibandingkan algoritme LC dan RT.

Pada Gambar 7.30 merupakan hasil dari pengujian dengan beban *memory usage* di kategori *low.* Algoritme *Response time* (RT) memiliki hasil memory usage yang paling rendah pada server 1 dan 2 dibandingkan dengan algoritme *Fuzzy* dan LC. Terjadinya hal ini, karena algoritme RT tidak menerapkan agen *Psutil* di setiap server yang menyebabkan menurunkan penggunaan *memory*. Sedangkan di server 3 *memory usage* paling kecil dengan nilai 60% dimiliki oleh algoritme fuzzy yang disebabkan server 3 tidak menerima beban trafik.

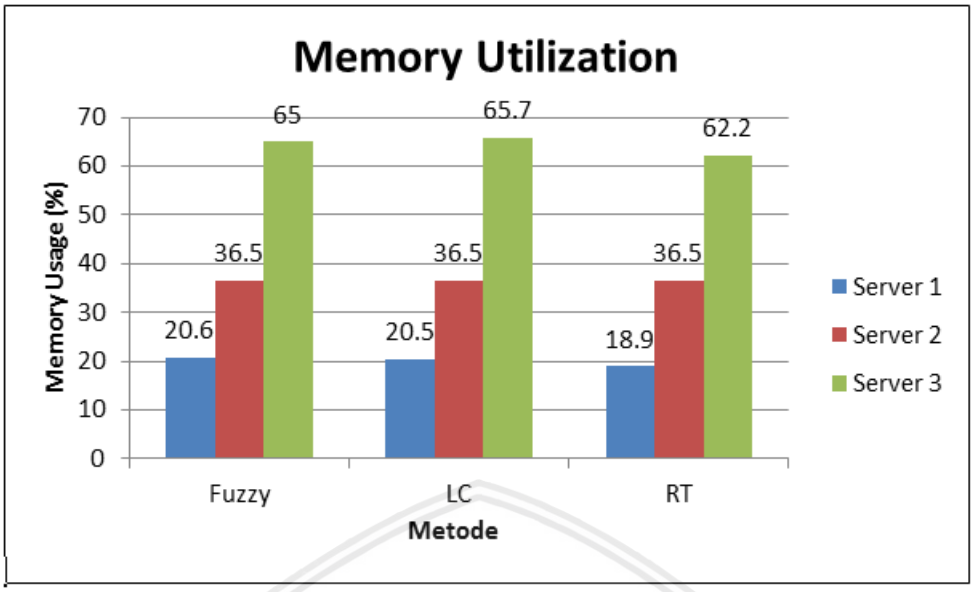


Gambar 7. 30 Grafik Memory Usage Pengujian Dengan Beban Skenario Low

Pada Gambar 7.31 merupakan grafik perbandingan pengujian kinerja dengan beban CPU *usage* di kategori *medium.* Algoritme *response time* (RT) memiliki hasil dari CPU *usage* paling rendah pada server 1 dengan nilai 30%. Hal ini disebabkan, server 1 pada algoritme RT mendapatkan pendistibusian trafik paling sedikit dibandingkan server 1 dengan algoritme *Fuzzy* dan LC. Kemudian CPU *usage* paling rendah diperoleh server 2 yang memiliki nilai 51 % dan pada server 3 yang memiliki nilai 52 % dimiliki oleh algoritme *Fuzzy*. Hal ini disebabkan, algoritme *Fuzzy* mendapatkan pendistibusian trafik pada server 2 dan 3 lebih sedikit dibandingkan algoritme LC dan RT.

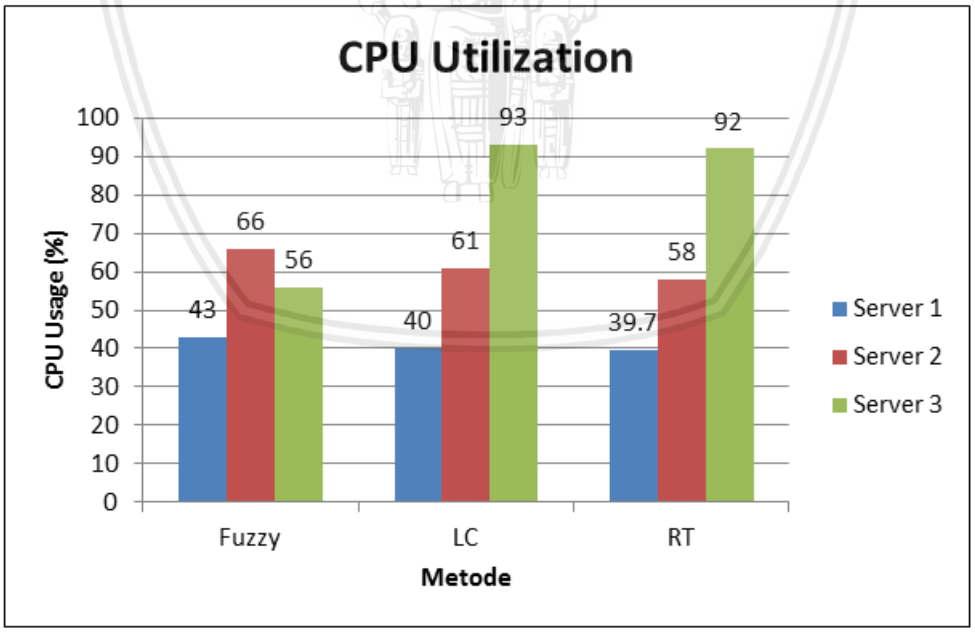


Gambar 7. 31 Grafik CPU Usage Pengujian Dengan Beban Skenario Medium

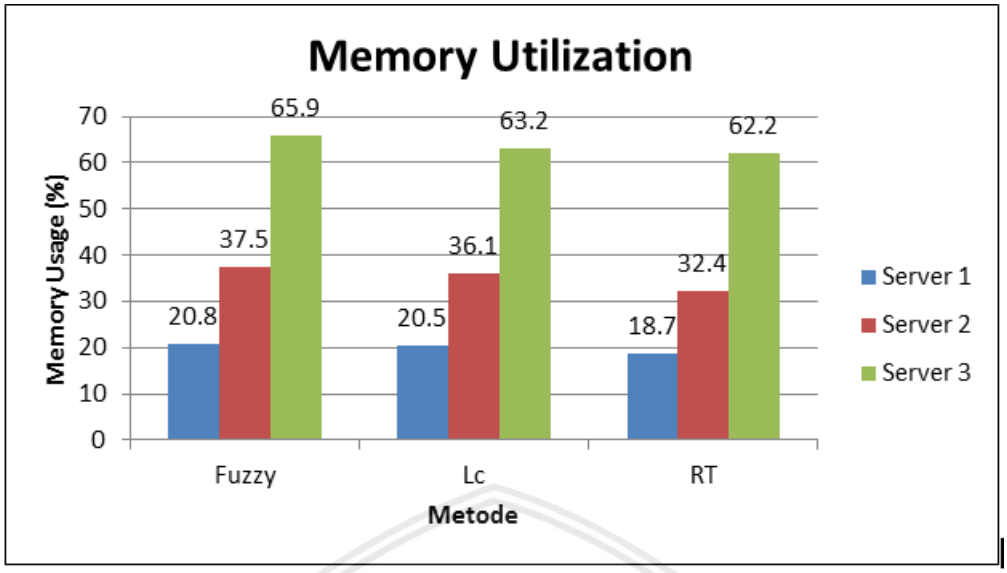


Gambar 7. 32 Grafik Memory Usage Pengujian Dengan Beban Skenario Medium

Pada Gambar 7.32 merupakan hasil dari pengujian dengan beban *memory usage* di kategori *medium.* Algoritme *Response time* (RT) memiliki hasil memory usage yang paling rendah pada seluruh server dibandingkan dengan algoritme *Fuzzy* dan LC. Terjadinya hal ini, karena algoritme RT tidak menerapkan agen *Psutil* di setiap server yang menyebabkan menurunkan penggunaan *memory*.



Gambar 7. 33 Grafik CPU Usage Pengujian Dengan Beban Skenario High



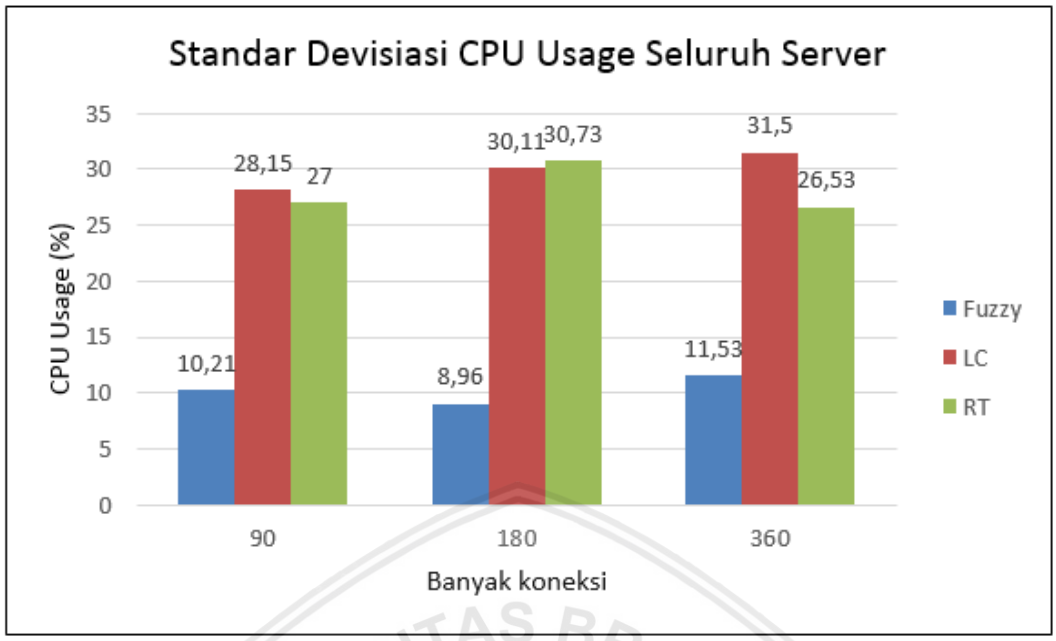
Gambar 7. 34 Grafik Memory Usage Pengujian Dengan Beban Skenario High

Berdasarkan Gambar 7.33 adalah grafik perbandingan kinerja CPU *usage* di kategori *high* pada pengujian dengan beban. Algoritme *response time* (RT) memiliki hasil dari CPU *usage* paling rendah pada server 1 dengan nilai 39,7% dan server 2 dengan nilai 58 %. Kemudian penggunaan CPU paling rendah pada server 3 dengan nilai 56 % dimiliki oleh algoritme *Fuzzy*. Hal ini disebabkan, algoritme *Fuzzy* mendapatkan pendistibusian trafik pada server 3 lebih sedikit dibandingkan algoritme LC dan RT.

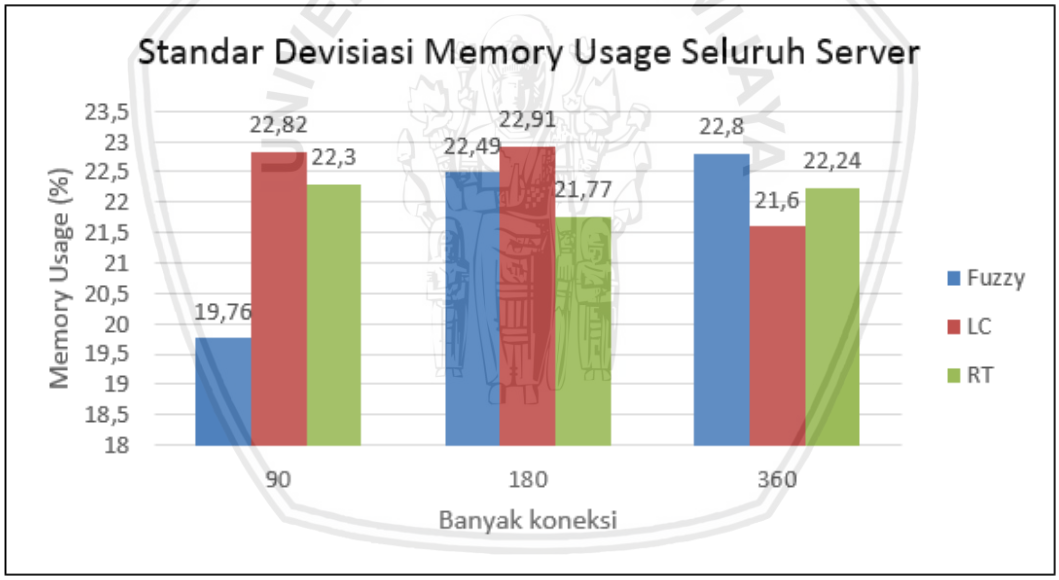
Pada Gambar 7.34 merupakan hasil dari pengujian dengan beban *memory usage* di kategori *high.* Algoritme *Response time* (RT) memiliki hasil memory usage yang paling rendah pada seluruh server dibandingkan dengan algoritme *Fuzzy* dan LC. Terjadinya hal ini, karena algoritme RT tidak menerapkan agen *Psutil* di setiap server yang menyebabkan menurunkan penggunaan *memory.*

Pada Gambar 7.35 menunjukan grafik standar deviasi dari CPU *Usage*. Metode fuzzy memperoleh nilai paling rendah serta menegaskan dapat mendistribusikan trafik secara seimbang berdasarkan penggunaan sumber daya server terkecil. CPU merupakan salah satu variabel dalam penentuan pendistribusian trafik menggunakan metode *Fuzzy.* Kemudian metode LC dan metode RT memiliki standart deviasi hampir sama. Hal ini dipengaruhi oleh pendistribusian trafik yang didasari *response* server yang paling cepat dan koneksi yang terkecil tanpa memperdulikan sumber daya server. Oleh karena itu, meyebabkan kondisi server yang memiliki sumber daya yang rendah dapat terbebani dibandingkan dengan jumlah sumber daya yang tinggi.

Pada Gambar 7.36 merupakan standar deviasi *Memory Usage.* Metode *response time* memiliki nilai yang paling rendah pada kategory 180. Kemudia metode Fuzzy memiliki nilai standar yang rendah pada kategori 90. Selanjutnya pada kategory 360 metode *Least Connection* memiliki nilaiystandar deviasiyyang rendah.



Gambar 7. 35 Grafik Standar Devisiasi CPU Usage Pengujian Dengan Beban

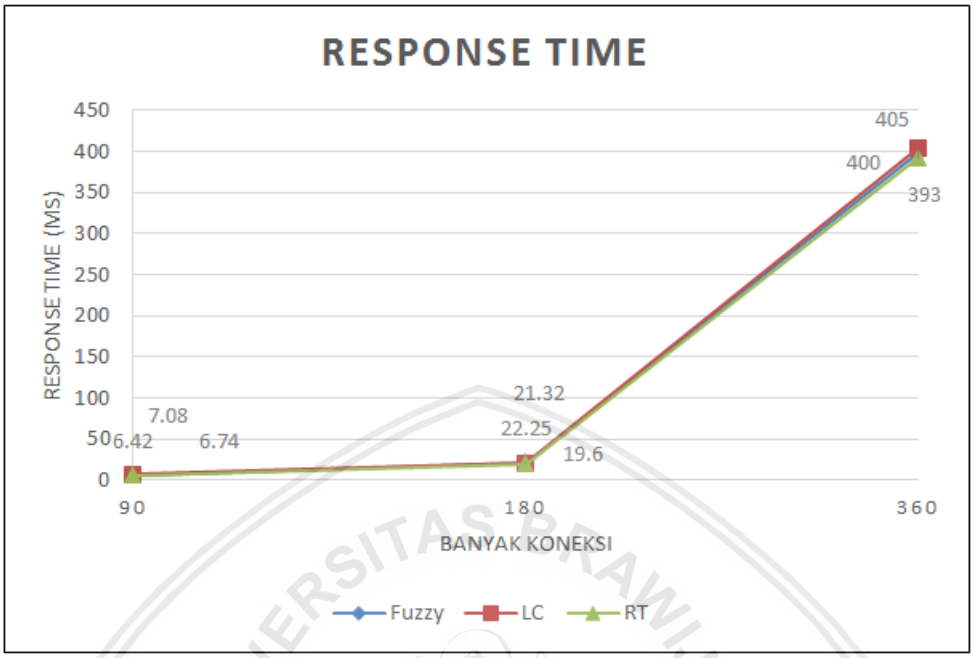


Gambar 7. 36 Grafik Standar Devisiasi Memory Usage Pengujian Dengan Beban

7.3.3 Pengujian Response Time

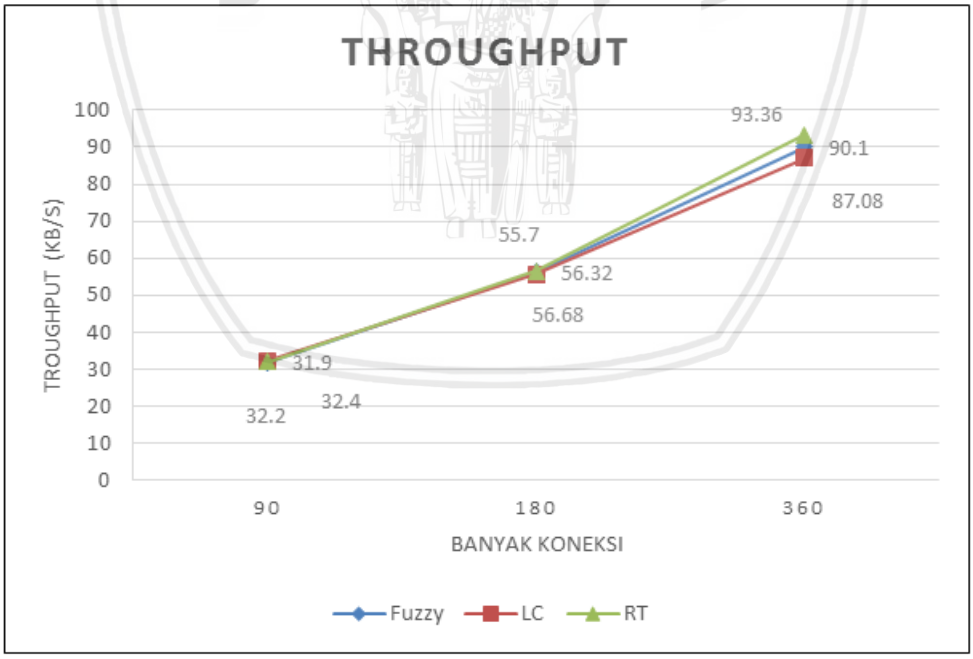
Pengujian dengan parameter *Response time* pada pengujian dengan beban yang bertujuan mendapatkan hasil *Response time* pada metode *Fuzzy* yang dibandingkan dengan algoritme LC dan RT. Pada setiap algoritme *Fuzzy*, LC dan RT mendapatkan hasil yang berbeda pada tiap skenario pengujian *90, 180* dan 360. Hal ini disebabkan semakin banyaknya *request* yang dilayani oleh masing-masing server, maka semakin banyak koneksi yang terjalin. Berdasarkan Gambar 7.37 algoritme RT lebih baik dibandingkan algoritme *Fuzzy* dan LC pada kategori *low* dengan nilai 6,42 ms, kategori *medium* dengan nilai 19,6 ms dan

kategori *high* dengan nilai 393 ms. Hal ini disebabkan metode response time membagi traffic berdasarkan *response time* tercepat dari setiap server.



Gambar 7. 37 Grafik Response Time Pengujian Dengan Beban

7.3.4 Pengujian Troughput



Gambar 7. 38 Grafik Troughtput Pengujian Dengan Beban

Dalam Pengujian *Troughput* pada pengujian tanpa beban yang bertujuan mengetahui hasil *Troughput* dari algoritme *Fuzzy* untuk dibandingkan dengan algoritme LC dan RT. Pada setiap algoritme *Fuzzy*, LC dan RT mendapatkan hasil yang berbeda pada tiap skenario pengujian *90, 180* dan 360. Hal ini disebabkan,

semakin banyaknya *request* dilayani oleh masing-masing server, maka nilai daro *Troughput* yang diperoleh akan bertambah tinggi. Berdasarkan Gambar 7.38 menjelaskan kategori *low* dengan nilai 32,4 KB/s, kategori *medium* dengan nilai 56,68 KB/s dan kategori *high* dengan nilai 93,36 dihasilkan oleh algoritme RT lebih tinggi dibandingkan algoritme *Fuzzy* dan LC. Hal ini dikarenakan semakin kecilnya *response time* maka semakin besar *troughput* yang dihasilkan.

BAB 8 PENUTUP

8.1 Kesimpulan

Setelah memalui seluruh proses penelitian, peneliti melakukan penarikan kesimpulan pada judul "Mekanisme Pembobotan Server Menggunakan Algoritme *Fuzzy* Pada Sistem Load Balancing Di Software Defined Network". Berdasarkan hasil pengujian tersebut, pengujian menggunakan beberapa parameter yang digunakan yaitu penggunaan CPU, penggunaan *memory*, distribusi trafik, *response time* dan *troughput*.

1. Penelitian ini menggunakan SDN Controller POX yang berperan sebagai pembagi beban dengan menggunakan algoritme *Fuzzy*. Selain itu diimplementasikan agen Psutil yang berfungsi untuk mengirimkan informasi kondisi server berupa sumber daya server. Agen tersebut, ditanamkan pada setiap server untuk membantu Controller mendistribusikan beban pada setiap server berdasarkan metode yang digunakan. Dari hasil implementasi, sistem tersebut dapat berjalan sesuai fungsi dari algoritme *Fuzzy* yang mendistribusikan beban berdasarkan penggunaan sumber daya server paling rendah.
2. Dari hasil pengujian kinerja tanpa beban yang dilakukan pada algoritme *Response Time* dan *Least Connection*, setiap server pada cluster tidak mengalami *overload* ketika proses pengiriman request pada skenario 90 req/s (*low)*. Sedangkan, di skenario 180 req/s (*medium*) dan 360 req/s (*high)*, server mengalami *overload* sehingga mempengaruhi *response time*, penggunaan sumber daya dan troughput khususnya pada server 2 dan 3.
3. Pada pengujian kinerja dengan beban yang menggunakan algoritme *Fuzzy*, di setiap kategori seluruh server tidak ada yang mengalami *overload* dan *down*. hal ini dikarenakan, algoritme *Fuzzy* mendistribusikan beban trafik berdasarkan penggunaan sumber daya server paling rendah. Sedangkan pada algoritme LC dan RT di kategori *medium* dan *high* server 2 dan 3 mengalami *overload* yang menyebabkan server *down*. hal ini dikarenakan, algoritme RT dan LC mendistribusikan beban trafik tanpa melihat kondisi server yang sebenarnya.
4. Dari hasil pengujian yang dilakukan, algoritme Fuzzy dan LC *response time* yang didapatkan lebih besar dan troughput yang rendah dibandingkan algoritme RT. Hal ini dikarenakan, kedua algoritme tersebut menggunakan Agen *Psutil* yang membebani kinerja dari setiap server. Sedangkan algoritme RT tidak menggunakan agen *Psutil.*

8.2 Saran

Bedasarkan penelitian yang sudah dilakukan, peneliti akan memuat sebuah saran dengan harapan dapat digunakan pada penelitian selanjutnya, sebagai berikut:

1. Dapat menggunakan *SDN Controller* selain POX seperti Ryu, Floodlight dan lain-lain.
2. Dapat melakukan metode pengujian dengan parameter yang berbeda.
3. Dapat menggunakan Web Server dengan spesifikasi yang lebih tinggi agar mendapatkan hasil yang lebih baik.

DAFTAR PUSTAKA

Psutil Documentation. (2018, 02 19). Retrieved September 1, 2018, from psutil documentation: http://psutil.readthedocs.io/en/latest/

Abdillah, M. (2018). Analisis Perbandingan Kinerja Metode Load Balancing Berbasis CPU Dengan Berbasis Response Time Pada Software Defined Network. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*.

Citrix. (2016). *Citrix XenServer Workload Balancing 6.5 Administrator's Guide.* Citrix inc.

Hu, F., Hao, Q., & Bao, K. (2014). A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. *IEEE COMMUNICATION SURVEYS & TUTORIALS, 6*.

Imam, S., Wijana, S., & Prawiti, W. H. (2010). PENERAPAN LOGIKA FUZZY PADA PENILAIAN MUTU SUSU SEGAR. *Jurnal Teknologi Pertanian , 11*.

Islahul Ardy, L. F. (2018). IMPLEMENTASI LOAD BALANCER BERDASARKAN SERVER STATUS PADA ARSITEKTUR SOFTWARE DEFINED NETWORK. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*.

Julianto, R. (2017). Implementasi Load balancing Menggunakan Metode Berbasis Sumber Daya CPU pada Software Defined Networking.

Karimi, A. (2009). A New Fuzzy Approach for Dynamic Load. *(IJCSIS) International Journal of Computer Science and Information Security, 6*.

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

LI, D. C., & Chang, F. M. (2007). *An In–Out Combined Dynamic Weighted Round-Robin Method for Network Load Balancing*.

McCauley, M. (2015, Maret 5). *POX Wiki*. Retrieved September 1, 2018, from https://openflow.stanford.edu/display/ONL/POX+Wiki

Mosbergen, D., & Jin, T. (1998). httperf — A Tool for Measuring Web Server Performance.

Open Networking Foundation. (2009). OpenFlow Switch Specification.

Paul, S., & Adhikari, M. (2018). Dynamic Load Balancing Strategy based on Resource Classification Technique in IaaS Cloud.

Rahmawan, H., & Gondokaryono, Y. S. (2009). *The Simulation of Static Load Balancing*.

Rahmawati, D. A. (2015). *PENERAPAN FUZZY LOGIC DENGAN MENGGUNAKAN METODE MAMDANI UNTUK*.

Rijayana, I. (2005). TEKNOLOGI LOAD BALANCING UNTUK MENGATASI BEBAN SERVER. *Seminar Nasional Aplikasi Teknologi Informasi*, 35.

Rosalia, M. (2016). IMPLEMENTASI HIGH AVAILABILITY SERVER MENGGUNAKAN METODE LOAD BALANCING DAN FAILOVER PADA VIRTUAL WEB SERVER CLUSTER. *e-Proceeding of Engineering, 3*, 4496.

Sabiya, & Singh, J. (2016). Weighted Round-Robin Load Balancing Using Software. *International Journal of Advanced Research in Computer Science and Software Engineering*.

Saelan, A. (2009). *LOGIKA FUZZY*.

Shaleh, A. (2015). IMPLEMENTASI METODE FUZZY MAMDANI DALAM MEMPREDIKSI TINGKAT KEBISINGAN LALU LINTAS. *Seminar Nasional Teknologi Informasi dan Multimedia 2015*.

Shin, M.-K., Nam, K.-H., & Kim, H.-J. (2012). Software Defined Network (SDN) : A Reference Architecture and Open APIs.

Wang, T., & Guo, X. (2017). A Fuzzy Synthetic Evaluation Algorithm with. *School of Computer and Communication Engineering*, 1035.

Wulandari, Y. (2011). *APLIKASI METODE MAMDANI DALAM PENENTUAN STATUS GIZI DENGAN INDEKS MASSA TUBUH (IMT) MENGGUNAKAN LOGIKA FUZZY*.