IMPLEMENTASI *LOAD BALANCING* MENGGUNAKAN ALGORITME *LEAST CONNECTION* DENGAN *AGEN PSUTILS* PADA *WEB SERVER*

SKRIPSI

Untuk memenuhi sebagian persyaratan
Memperoleh gelar Sarjana Komputer

Disusun oleh:
Muhammad Sholeh
NIM: 135150201111276

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

PENGESAHAN

IMPLEMENTASI *LOAD BALANCING* MENGGUNAKAN ALGORITME *LEAST CONNECTION* DENGAN *AGEN PSUTILS* PADA *WEB SERVER*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Muhammad Sholeh
NIM: 135150201111276

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

Dosen Pembimbing I	Dosen Pembimbing II
Widhi Yahya, S.Kom., M.Sc.	Ir. Primantara Hari Trisnawan, M. Sc.
NIK: 2016078911211001	NIP: 19680912 199403 1 002

Mengetahui
Ketua Jurusan Teknik Informatika

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

PERNYATAAN ORISINALITAS

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

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

Malang, 02 Agustus 2018

Muhammad Sholeh

NIM: 135150201111276

KATA PENGANTAR

Segala puji dan syukur penulis ucapkan atas kehadirat Allah SWT yang telah memberikan Karunia dan Rahmat-Nya sehingga penulis mampu menyelesaikan penelitian ini dengan judul "*Implementasi Load Balancing Menggunakan Algoritme Least Connection dengan Agen Psutils pada Web Server*". Penelitian ini dilakukan untuk memenuhi syarat untuk mendapatkan gelar sarjana di Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya. Selain itu penelitian ini juga dapat terselesaikan atas bantuan, dukungan serta bimbingan dari berbagai pihak. Untuk itu penulis ingin mengucapkan banyak terima kasih kepada:

1. Yang terhormat bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya.
2. Yang terhormat bapak Tri Astoto Kurniawan , S.T.,M. T,Ph. D selaku ketua jurusan Teknik Informatika.
3. Yang terhormat bapak Widhi Yahya S.Kom., M.Sc sebagai dosen pembimbing 1 yang telah banyak meluangkan waktunya untuk memberikan ilmu dan saran kepada penulis terkait pengerjaan penelitian ini.
4. Yang terhormat bapak Ir. Primantara Hari Trisnawan, M. Sc sebagai dosen pembimbing 2 yang sangat sabar memberikan arahan dan bimbingan kepada penulis dalam proses penelitian.
5. Seluruh dosen Fakultas Ilmu Komputer Universitas Brawijaya yang telah mengajarkan ilmu pengetahuan dan memberikan motivasi selama menjalani perkuliahan.
6. Orang tua penulis yang selalu mempertanyakan proses pengerjaan skripsi ini sehingga penulis selalu sadar untuk menyelesaikan penelitian ini.
7. Teman-teman khususnya Muharrom Abdillah dan Hasbi Razzak, yang selalu memberikan bantuan, saran dan motivasi kepada penulis dalam pengerjaan skripsi.
8. Teman-teman seperjuangan lain yang tidak dapat penulis sebutkan satu persatu disini atas seluruh bantuan, saran dan motivasi yang telah diberikan.

Terakhir, penulis sangat berharap semoga seluruh kebaikan mereka mendapatkan balasan dari Allah SWT. Penulis menyadari bahwa skripsi ini jauh dari kesempurnaan, oleh karena itu penulis mengharapkan saran dan kritik untuk pengembangan skripsi ini selanjutnya agar mencapai hasil yang lebih baik dan dapat berkontribusi memberikan pengetahuan khususnya pada bidang Ilmu Komputer. Semoga penelitian yang dikerjakan oleh penulis ini dapat bermanfaat

bagi pembaca khususnya teman-teman mahasiswa Fakultas Ilmu Komputer Universitas Brawijaya.

Malang, 02 Agustus 2018

Penulis

avsholeh@gmail.com

ABSTRAK

Media sosial dan situs berita merupakan layanan di Internet yang paling banyak dikunjungi oleh pengguna Internet. Sehingga layanan itu dituntut untuk mampu menangani trafik yang sangat tinggi. Jika layanan tersebut menggunakan *web server* tunggal (*single server*) maka sewaktu-waktu dapat menyebabkan *Single Point of Failure. Single Point of Failure* adalah kesalahan yang terjadi pada sistem yang menyebabkan berhenti bekerja. Solusi yang dapat menangani permasalahan tersebut adalah dengan menggunakan metode *load balancing*.

Load balancing adalah suatu metode yang dapat membagi trafik secara seimbang ke beberapa server. *Load balancing* menggunakan algoritme untuk mendistribusikan trafik ke beberapa server. Algoritme yang digunakan antara lain Random, Round Robin, Least Loaded dan Least Connection. Pada penelitian ini akan dilakukan suatu pengembangan terhadap algoritme *Least Connection* untuk *load balancing* pada arsitektur *Software Defined Network* (SDN). *Least Connection* adalah algoritme *load balancing* yang akan mengarahkan trafik ke *web server* yang memiliki koneksi paling sedikit. Penelitian ini menggunakan sebuah agen yang disebut dengan *Agen Psutils.* Agen ini akan mengirimkan informasi mengenai jumlah koneksi aktif yang dimiliki oleh *web server* ke *SDN Controller*. Dari informasi tersebut, *SDN Controller* dapat mengarahkan trafik ke *web server* yang memiliki koneksi paling sedikit.

Hasil dari penelitian menunjukkan bahwa algoritme *Least Connection* dengan *Agen Psutils* dapat mendistribusikan trafik berdasarkan jumlah koneksi aktif yang dimiliki oleh *server*. Kemudian, ketika melakukan perbandingan antara algoritme *Least Connection* berbasis *Agen Psutils* (*LC Agen Psutils*) dengan *Least Connection* berbasis *Expired Flow* (*LC Flow*). *LC Agen Psutils* dapat mengirimkan data berukuran 1,2 GigaBytes sedangkan *LC Flow* tidak dapat menyelesaikan proses pengiriman data. Pada pengiriman *request* berjumlah 400, *LC Agen Psutils* memiliki *response time* yang lebih kecil dibandingkan *Round Robin* dan *LC Flow*. Dimana *response time* dari LC Agen Psutils adalah 242,51 ms, sedangkan *Round Robin* dan *LC Flow* adalah 261,61 ms dan 279,81 ms.

Kata kunci: *load balancing, Software Defined Network, Least Connections, POX controller*

ABSTRACT

Social media and news sites are services on the Internet that are most visited by Internet users. So, that service is required to be able to handle very high traffic. If the service uses a single web server, it can cause a Single Point of Failure. Single Point of Failure is an error that occurs in the system that causes it to stop working. The solution that can handle these problems is to use a method of load balancing.

Load balancing is a method that can distribute traffic to multiple servers. Load balancing uses algorithms to distribute traffic to multiple servers. The algorithms include Random, Round Robin, Least Loaded and Least Connection. This research develop the Least Connection algorithm for load balancing in the Software Defined Network (SDN). Least Connection is a load balancing algorithm that will direct traffic to web servers that have the least connection. This research uses an agent called Agent Psutils. This agent send information to SDN Controller about the number of active connections of web servers. From this information, SDN Controller can direct traffic to web servers that have the least connection.

The results of the study show that the Least Connection algorithm with Agent Psutils can distribute traffic based on the number of active connections on server. Then, when comparing the Least Connection algorithm based on Agent Psutils (LC Agent Psutils) with Least Connection based on Expired Flow (LC Flow). LC Agent Psutils can send data measuring 1.2 GigaBytes while LC Flow cannot complete the data transmission process. At 400 requests, LC Agent Psutils have a smaller response time than Round Robin and LC Flow. Where the response time of the LC Agent Psutils is 242.51 ms, while the Round Robin and LC Flow are 261.61 ms and 279.81 ms.

Keywords: load balancing, Software Defined Network, Least Connections, POX controller

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
KATA PENGANTAR iv
ABSTRAK vi
ABSTRACT vii
DAFTAR ISI viii
DAFTAR TABEL xi
DAFTAR GAMBAR xii
DAFTAR LAMPIRAN xiv
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Rumusan masalah 2
1.3 Tujuan 2
1.4 Manfaat 2
1.5 Batasan masalah 3
1.6 Sistematika pembahasan 3
BAB 2 LANDASAN KEPUSTAKAAN 5
2.1 Kajian Pustaka 5
2.2 *Load balancing* 7
2.3 *Software Defined Network* (SDN) 8
2.3.1 Open*Flow* 10
2.3.2 SDN *Controller* 10
2.4 *Web Server* 10
2.5 Mininet (SDN Network Simulator) 11
2.6 Psutils 11
2.7 Apache Jmeter 11
BAB 3 METODOLOGI 13
3.1 Studi Literatur 13
3.2 Rekayasa Kebutuhan 14
3.3 Perancangan dan Implementasi 14

3.3.1 Perancangan 14
3.3.2 Implementasi 14
3.4 Pengujian 14
3.5 Kesimpulan 15
BAB 4 REKAYASA KEBUTUHAN 16
4.1 Deskripsi Umum Sistem 16
4.2 Deskripsi Kebutuhan Sistem 17
4.2.1 Kebutuhan Fungsional 17
4.2.2 Kebutuhan Non-Fungsional 18
4.2.3 Kebutuhan Lingkungan Kerja Sistem 19
BAB 5 PERANCANGAN DAN IMPLEMENTASI 20
5.1 Perancangan 20
5.1.1 Perancangan Arsitektur SDN 20
5.1.2 Perancangan Sistem *Load balancing* 21
5.1.3 Perancangan Algoritme *Least Connection* 25
5.1.4 Perancangan Algoritme Pada *Agen Psutils* 26
5.1.5 Perancangan *Client* 27
5.2 Implementasi 27
5.2.1 Implementasi Arsitektur SDN 27
5.2.2 Implementasi Sistem *Load balancing* 29
5.2.3 Implementasi Algoritme *Least Connection* 31
5.2.4 Implementasi Algoritme Pada *Agen Psutils* 32
5.2.5 Implementasi *Client* 34
BAB 6 PENGUJIAN 35
6.1 Pengujian *Download* 35
6.1.1 Metode Berbasis *Agen Psutils* 35
6.1.2 Metode Berbasis *Flow* 36
6.1.3 Pembahasan 38
6.2 Pengujian Kinerja 38
6.2.1 Skenario Pengiriman 100 *Request* 39
6.2.2 Skenario Pengiriman 200 *Request* 42
6.2.3 Skenario Pengiriman 400 *Request* 46

6.2.4 Pembahasan 51
BAB 7 PENUTUP 52
7.1 Kesimpulan 52
7.2 Saran 53
DAFTAR PUSTAKA 54
LAMPIRAN DATA HASIL PENGUJIAN 55
7.1 Skenario Pengiriman 100 *Request* 55
7.1.1 Distribusi Trafik 55
7.1.2 CPU Utilization 55
7.1.3 Memory Utilization 55
7.1.4 Response Time 55
7.2 Skenario Pengiriman 200 *Request* 56
7.2.1 Distribusi Trafik 56
7.2.2 CPU Utilization 56
7.2.3 Memory Utilization 56
7.2.4 Response Time 56
7.3 Skenario Pengiriman 400 *Request* 57
7.3.1 Distribusi Trafik 57
7.3.2 CPU Utilization 57
7.3.3 Memory Utilization 57
7.3.4 Response Time 57

DAFTAR TABEL

Tabel 2.1 Kajian Pustaka 6
Tabel 5.1 Pseudocode Algoritme *Least Connection* 25
Tabel 5.3 Pseudocode Algoritme pada *Agen Psutils* 26
Tabel 5.4 Kode sumber Algoritme *Least Connection* 31
Tabel 5.5 Kode sumber *Agen Psutils* 32

DAFTAR GAMBAR

Gambar 2.1 Sistem Tanpa *Load balancing* 8
Gambar 2.2 Sistem Dengan *Load balancing* 8
Gambar 2.3 Arsitektur *Software Defined Network* 9
Gambar 2.4 *Openflow Controller* dan *Openflow Switch* 10
Gambar 3.1 Diagram Alir Metodologi Penelitian 13
Gambar 5.1 Arsitektur *Software Defined Network* 20
Gambar 5.2 Alur Sistem Secara Umum 22
Gambar 5.3 Alur Pemeriksaan Status *Server* 23
Gambar 5.4 Alur Pemeriksaan Paket dari *Client* 23
Gambar 5.5 Alur Pemeriksaan Paket dari *Server* 24
Gambar 5.6 Alur Pengiriman Jumlah Koneksi 25
Gambar 5.7 Mininet GUI (Miniedit) 28
Gambar 5.8 Topologi Pada Mininet GUI (Miniedit) 28
Gambar 5.9 Tampilan Ketika POX *Controller* Dijalankan 30
Gambar 5.10 Tampilan Terminal Pada *Server h1* 31
Gambar 5.11 Tampilan Terminal Pada *Server h1* Ketika Mendapatkan *Request* . 31
Gambar 6.1 *Client* Mengakses Data Pada *Server* 35
Gambar 6.2 *Controller* Mengarahkan Trafik ke *Server h1* 35
Gambar 6.3 *Server h1* Melayani *Request* dari *Client* 36
Gambar 6.4 *Request* data dari sisi *Client* Telah Selesai 36
Gambar 6.5 *Client* Mengakses Data Pada *Server* 36
Gambar 6.6 *Controller* Mengarahkan Trafik ke *Server h1* 37
Gambar 6.7 *Server h1* Melayani *Request* dari *Client* 37
Gambar 6.8 *Request Data* dari sisi *Client* Terhenti 37
Gambar 6.9 Distribusi Trafik Pengiriman 100 *Request* 39
Gambar 6.10 CPU Utilization Pengiriman 100 *Request* 40
Gambar 6.11 Memory Utilization Pengiriman 100 *Request* 40
Gambar 6.12 Response Time HTML Pengiriman 100 *Request* 41
Gambar 6.13 Response Time Image Pengiriman 100 Request 42
Gambar 6.15 Distribusi Trafik Pengiriman 200 *Request* 43

Gambar 6.16 CPU Utilization Pengiriman 200 *Request* 43
Gambar 6.17 Memory Utilization Pengiriman 200 *Request* 44
Gambar 6.18 Response Time HTML Pengiriman 200 *Request* 45
Gambar 6.19 Response Time Image Pengiriman 200 *Request* 45
Gambar 6.20 Distribusi Trafik Pengiriman 400 *Request* 46
Gambar 6.21 Distribusi Trafik HTML 400 *Request* 47
Gambar 6.22 Distribusi Trafik Image 400 *Request* 47
Gambar 6.23 CPU Utilization Pengiriman 400 *Request* 48
Gambar 6.24 Memory Utilization Pengiriman 400 *Request* 49
Gambar 6.25 Response Time HTML Pengiriman 400 *Request* 50
Gambar 6.26 Response Time Image Pengiriman 400 *Request* 50

DAFTAR LAMPIRAN

LAMPIRAN DATA HASIL PENGUJIAN…… 69

BAB 1 PENDAHULUAN

1.1 Latar belakang

Media sosial dan situs berita merupakan layanan di Internet yang paling banyak dikunjungi oleh pengguna Internet. Sehingga layanan itu dituntut untuk mampu menangani trafik yang sangat tinggi. Jika layanan tersebut menggunakan *web server* tunggal (*single server*) maka sewaktu-waktu dapat menyebabkan *Single Point of Failure. Single Point of Failure* adalah kesalahan yang terdapat pada sistem yang menyebabkan sistem berhenti bekerja (Dooley, 2002). Solusi yang dapat menangani permasalahan tersebut adalah dengan menggunakan metode *load balancing*.

Load balancing adalah suatu metode yang dapat membagi trafik secara seimbang ke beberapa server. Sehingga jika salah satu server tidak mampu menerima trafik dari pengguna, maka trafik tersebut dapat dilayani oleh server lain. Untuk membagikan trafik kepada beberapa *server*, *load balancer* menggunakan algoritme untuk menentukan *server* yang tepat untuk melayani trafik tersebut. Adapun beberapa algoritme yang saat ini diterapkan untuk *load balancing* antara lain *Random, Round-Robin, Weighted Round Robin, Least Connection, Least Loaded Server* dll (Mahmood & Rashid, 2011).

Arsitektur *Software Defined Network* (SDN) merupakan paradigma baru dalam jaringan yang menawarkan pengembangan terhadap teknik *load balancing*. *Software Defined Network* (SDN) ini bersifat *programmable* (dapat diprogram), sehingga para *Network Administrator* dapat merancang dan mengimplementasikan strategi dan algoritme *load balancing* tanpa keterbatasan *vendor* (Kaur & Singh, 2015). Dari kemampuan tersebut, banyak penelitian yang telah dilakukan untuk menerapkan *load balancing* pada *Software Defined Network* (SDN) salah satunya adalah penelitian yang dilakukan oleh (Erine, 2016) yang berjudul "*Analisis Openflow Load balancing Webserver dengan Algoritme Least Connection Pada Software Defined Network*". Algoritme *Least Connection* merupakan algoritme penjadwalan yang dinamis dan dapat digunakan untuk *load balancing*. Algoritme ini dapat mengarahkan secara langsung koneksi pada jaringan ke *server* yang memiliki jumlah koneksi yang paling sedikit (Mustafa, 2015). Untuk menerapkan algoritme *Least Connection* diperlukan cara untuk menentukan jumlah koneksi yang dimiliki oleh *server*. Untuk mengetahui jumlah koneksi yang dimiliki oleh *server*, penelitian tersebut menggunakan informasi *flow* yang tersimpan pada perangkat *Switch*. *Flow* merupakan aliran data yang terdapat pada jaringan. Aliran data tersebut tersimpan di *flow table* pada perangkat *Switch* yang mendukung protokol *openflow*. Namun, *flow* pada *openflow* memiliki batas waktu (*timeout*) untuk menentukan kapan *flow* tersebut dihapus. Jika koneksi antara *Client* dan *server* masih berlangsung dan *flow* sudah mencapai batas waktunya maka *flow* tersebut akan dihapus dari *flow table*. Sehingga akan menimbulkan kesalahan pada informasi mengenai berapa jumlah

koneksi yang dimiliki oleh *server* sebenarnya. Selain itu juga menyebabkan terjadinya permasalahan pada koneksi, jika koneksi tersebut berlangsung melebihi batas waktu *timeout.*

Untuk itu penelitian ini ingin mengatasi permasalahan tersebut dengan mengimplementasikan metode untuk mengetahui jumlah beban koneksi yang dimiliki oleh *server* dengan menggunakan *Agen Psutils* serta mengatasi permasalahan yang terjadi pada koneksi yang berlangsung lama. *Agen Psutils* ini berfungsi untuk mengambil informasi tentang berapa jumlah koneksi yang sedang terhubung pada *server* serta mengirimkan informasi tersebut kepada *Controller*. Kemudian pada penelitian ini juga akan dilakukan perbandingan antara metode yang menggunakan *Agen Psutils* dengan metode yang menggunakan *Expired Flow*. Dengan melakukan perbandingan ini, kita mengetahui metode terbaik yang dapat digunakan pada implementasi algoritme *Least Connection* untuk *load balancing* khususnya pada arsitektur *Software Defined Network*.

1.2 Rumusan masalah

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

1. Bagaimana cara mengimplementasikan algoritme *Least Connection* dengan *Agen Psutils* pada arsitektur *Software Defined Network*?
2. Bagaimana cara melakukan pengujian terhadap sistem *load balancing* menggunakan algoritme *Least Connection* dengan *Agen Psutils* yang diterapkan pada arsitektur *Software Defined Network?*
3. Bagaimana cara membandingkan kinerja antara algoritme *Least Connection* yang menggunakan *Agen Psutils*, algoritme *Least Connection* yang menggunakan *Expired Flow* dan algoritme *Round Robin*?

1.3 Tujuan

Adapun tujuan dari penelitian ini adalah sebagai berikut:

1. Mengimplementasikan algoritme *Least Connection* dengan *Agen Psutils* ke dalam sistem *load balancing* pada arsitektur *Software Defined Network*.
2. Menguji kinerja dari sistem *load balancing* yang menggunakan algoritme *Least Connection* dengan *Agen Psutils* pada arsitektur *Software Defined Network*.
3. Membandingan kinerja dari algoritme *Least Connection* dengan *Agen Psutils*, *Least Connection* dengan *Expired Flow* dan *Round Robin*.

1.4 Manfaat

Berikut ini adalah manfaat yang dapat diperoleh dari penelitian ini yaitu:

1. Dapat mengetahui bagaimana cara mengimplementasikan sistem *load balancing* menggunakan algoritme *Least Connection* dengan *Agen Psutils* pada *Software Defined Network*.
2. Dapat mengetahui perbandingan kinerja antara metode *load balancing* dengan algoritme *Least Connection* berbasis *Agen Psutils* dan berbasis *flow*.
3. Dapat dijadikan sebagai acuan untuk penggunaan algoritme yang efisien pada *load balancing* di *Software Defined Network* (SDN).

1.5 Batasan masalah

Agar permasalahan pada penelitian ini dapat terfokus, maka ditentukanlah beberapa batasan antara lain:

1. Penelitian ini menggunakan teknologi virtualisasi jaringan untuk penerapan sistem *load balancing* pada arsitektur *Software Defined Network*.
2. Sistem *load balancing* hanya mendukung layanan yang menggunakan *Hypertext Transfer Protocol* (HTTP).
3. Data hasil pengujian yang digunakan merupakan data yang didapatkan dari lingkungan kerja peneliti.

1.6 Sistematika pembahasan

Berikut ini adalah struktur dari penelitian ini yang akan dibagi menjadi beberapa bagian yaitu:

BAB I PENDAHULUAN

Pada tahap ini akan dibahas latar belakang dari penelitian. Berawal dari masalah serta solusi yang dapat diraih. Kemudian dilanjutkan dengan rumusan masalah yaitu pertanyaan mengenai hal-hal apa saja yang akan diteliti. Selanjutnya tujuan dan manfaat penelitian serta batasan masalah.

BAB II LANDASAN KEPUSTAKAAN

Pada tahap ini akan diuraikan kajian pustaka dan dasar teori yang mendasari sistem *load balancing* di *web server* menggunakan algoritme *Least Connection* dengan *Agen Psutils* pada *Software Defined Network*.

BAB III METODOLOGI

Pada bagian ini akan dijelaskan langkah-langkah yang akan dilakukan pada penelitian meliputi: studi literatur, rekayasa kebutuhan, perancangan dan implementasi, pengujian serta pengambilan kesimpulan.

BAB IV REKAYASA KEBUTUHAN

Pada bagian ini akan dibahas tentang kebutuhan yang diperlukan dalam perancangan dan implementasi terhadap sistem.

BAB V PERANCANGAN DAN IMPLEMENTASI

Pada tahap ini akan dijelaskan tentang bagaimana merancang dan mengimplementasikan sistem *load balancing* di *web server* menggunakan algoritme *Least Connection* dengan *Agen Psutils* pada *Software Defined Network*.

BAB VI PENGUJIAN

Pada bab ini akan diuraikan tentang bagaimana melakukan pengujian terhadap sistem *Load Balancing* di *Web Server* menggunakan algoritme *Least Connection* dengan *Agen Psutils* pada *Software Defined Network*.

BAB VII PENUTUP

Pada bab ini akan dipaparkan tentang kesimpulan yang diperoleh dari hasil pengujian dan analisis serta saran untuk penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Kajian pustaka pada penelitian ini membahas tentang penelitian sebelumnya yang akan menjadi acuan dalam penelitian ini. Selain itu juga peneliti juga akan membahas objek maupun metode yang digunakan dari penelitian sebelumnya sebagai perbandingan dengan penelitian ini.

Penelitian sebelumnya yang berjudul "*Robin Based Load balancing in Software Defined Networking*" menerapkan *Load Balancing* di *Web Server* pada arsitektur *Software Defined Network* dengan menggunakan algoritme *Round Robin*. Pada penelitian ini beban didistribusikan secara terurut tanpa memperhatikan sumber daya pada *server*. Namun, hasil penelitian ini dapat bekerja dengan baik pada arsitektur *Software Defined Network* dan mempunyai kinerja yang lebih baik dibandingkan algorima *random* (Kaur & Singh, 2015). Pada penelitian tersebut menggunakan POX sebagai SDN *Controller*. Oleh karena itu, pada penelitian ini juga akan menggunakan SDN *Controller* yang sama untuk memenuhi kebutuhan sistem *load balancing*.

Kemudian adapun penelitian yang berjudul "*Analisis Openflow Load balancing Webserver dengan Algoritme Least Connection Pada Software Defined Network".* Penelitian tersebut mencoba menerapkan algoritme *Least Connection* pada arsitektur *Software Defined Network*. Penelitian tersebut juga menggunakan POX sebagai SDN *Controller* dan dapat berjalan dengan baik. Namun, pada implementasinya terdapat perbedaan pada penerapan algoritme *Least Connection*. Penelitian tersebut menggunakan *expired flow* sebagai penentu jumlah koneksi yang dimiliki oleh masing-masing *server* pada cluster (Erine, 2016). Hasilnya penelitian tersebut dapat berjalan dengan baik pada arsitektur *Software Defined Network*. Namun, secara kinerja belum dapat dipastikan karena pada penelitian tersebut tidak dilakukan perbandingan terhadap algoritme yang lain. Oleh karena itu, penelitian yang akan dilakukan dapat memperbaiki kekurangan dari penelitian sebelumnya.

Selanjutnya pada penelitian berjudul "*Implementasi Load Balancing Menggunakan Metode Berbasis Sumber Daya CPU pada Software Defined Networking*" menggunakan sebuah agen yang berfungsi untuk mengirimkan penggunaan CPU pada masing-masing *server* didalam cluster. Penelitian tersebut dapat membagi trafik secara seimbang berdasarkan sumber daya CPU yang dimiliki oleh server. Selain itu, agen yang digunakan dapat berjalan sesuai dengan fungsionalitasnya. Oleh karena itu, penelitian ini akan menggunakan konsep agen yang sama untuk mengirimkan jumlah koneksi pada *server*.

Pada penelitian lain yang berjudul "*Load balancing Algorithms Round-Robin, LeastConnection And Least Loaded Efficiency"* mencoba menguji tingkat efisiensi antara sistem *load balancing* dengan algoritme *Round Robin*, *Least Connection* dan *Least Loaded*. Pengujian tersebut menggunakan parameter *CPU*

Utilization dan jumlah distribusi trafik HTTP pada masing-masing *server*. Hasilnya algoritme Least Loaded lebih banyak menggunakan CPU jika dibandingkan dengan algoritme *Round Robin* dan *Least Connection* (Mustafa, 2015). Parameter pengujian tersebut dapat menjadi pertimbangan dalam penelitian ini untuk digunakan pada tahap pengujian.

Tabel 2.1 merupakan perbandingan dari penelitian terdahulu dengan penelitian yang akan dilakukan:

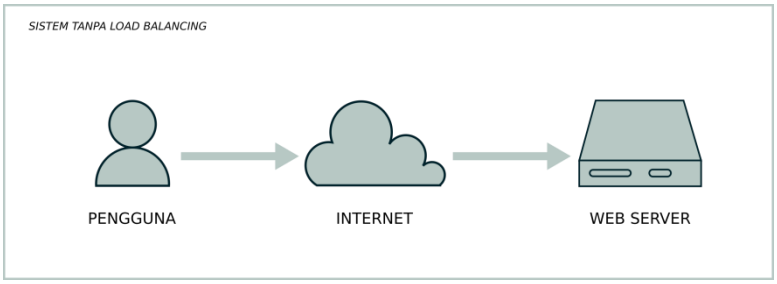
Tabel 2.1 Kajian Pustaka

No	Nama Peneliti, Tahun, dan Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1	Kaur & Singh. 2015. Round-Robin Based *Load balancing* in *Software Defined Network*ing. SBS State Technical Campus.	Mengimplementasikan sistem *load balancing* dengan arsitektur *Software Defined Network* dan menggunakan library POX sebagai *Controller*.	Mengimplementasikan sistem *load balancing* dengan menggunakan algoritme *Round Robin*.	Mengimplementasikan sistem *load balancing* dengan menggunakan algoritme *Least Connection*.
2	Erine, K. 2016. *Analisis Openflow Load balancing Webserver dengan Algoritme Least Connection Pada Software Defined Network*. Universitas Brawijaya, Malang.	Mengimplementasikan sistem *load balancing* dengan arsitektur *Software Defined Network* dan menggunakan library POX sebagai *Controller*.	Mengimplementasikan algoritme *Least Connection* pada *load balancing* dengan menggunakan *flow* sebagai parameter penentu jumlah koneksi.	Mengimplementasikan algoritme *Least Connection* dengan menggunakan metode *Agen Psutils* untuk mendapatkan jumlah koneksi yang dimiliki oleh *server*.
3	Julianto, R. 2016. Implementasi	Mengimplementasikan sistem load balancing	Menggunakan *Agen Psutils* untuk	Menggunakan *Agen Psutils* untuk

	Load Balancing Menggunakan Metode Berbasis Sumber Daya CPU pada *Software Defined Networking.* Universitas Brawijaya Malang.	dengan arsitektur Software Defined Network dan menggunakan library POX sebagai Controller.	mengetahui penggunaan CPU pada server didalam cluster.	mengetahui jumlah koneksi yang terdapat pada server didalam cluster.
4	Mustafa, E & Amin Mubark, I. 2015. *Load balancing Algorithms Round-Robin, LeastConnection And Least Loaded Efficiency*. International Journal Of Computer And Information Technology.	Menggunakan parameter distribusi trafik dan CPU Utilization	Membandingkan algoritme *Round Robin*, *Least Connection* dan least loaded efficiency dengan menggunakan simulator opnet.	Mengimplementasikan algoritme *Least Connection* dengan menggunakan mininet sebagai simulator

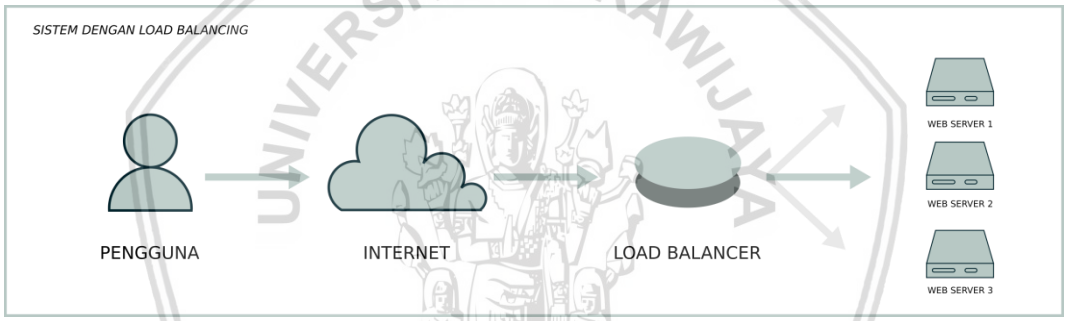
2.2 *Load balancing*

Load balancing merupakan teknik yang digunakan untuk mendistribusikan beban atau muatan pada beberapa *server*. Dengan menggunakan teknik ini kita dapat mengoptimalkan sumber daya, *response time* dan *throughput* karena memiliki lebih dari satu *server* yang dapat dijadikan cadangan jika salah satu *server* mengalami down (Lukitasari, 2010).



Gambar 2.1 Sistem Tanpa *Load balancing*

Gambar 2.1 merupakan gambaran tentang layanan web yang tidak menggunakan sistem *load balancing*. Setiap permintaan pengguna terhadap layanan pada *Web Server* akan dilayani oleh satu *Web Server* saja. Jika *Web Server* tersebut mengalami *down* maka pengguna tidak dapat mengakses layanan dari *Web Server* tersebut. Selain itu, jika terlalu banyak pengguna yang mengakses layanan dan *Web Server* tersebut tidak mampu menanganinya maka akan mempengaruhi kinerja dari *Web Server* tersebut.



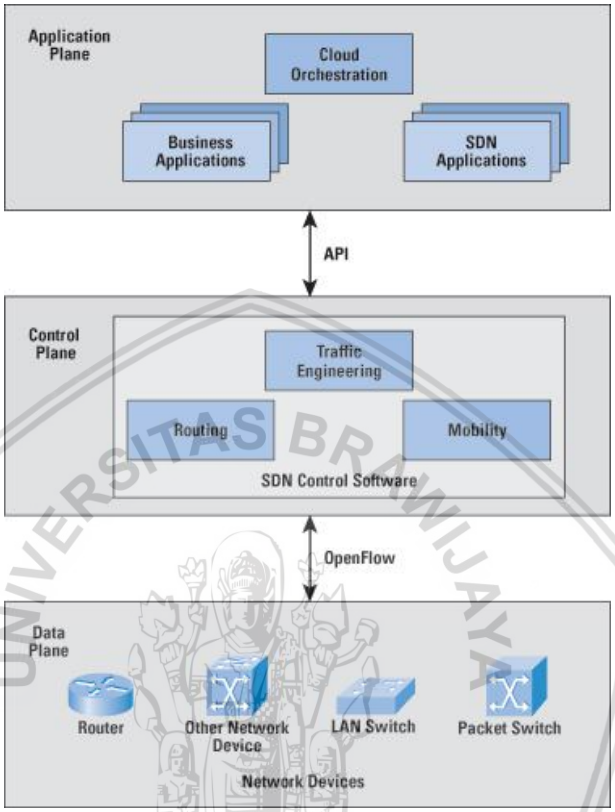
Gambar 2.2 Sistem Dengan *Load balancing*

Gambar 2.2 merupakan gambaran tentang layanan web yang menggunakan sistem *load balancing*. Terdapat 1 *load balancer* dan 3 *Web Server* yang dapat melayani permintaan dari pengguna. *Load balancer* akan mengatur trafik yang datang dan menentukan *Web Server* mana yang akan melayani trafik tersebut berdasarkan algoritme yang digunakan. Sebagai contoh, jika load balancer menggunakan algoritme *Least Connection* maka load balancer akan memilih *server* yang memiliki jumlah koneksi yang paling sedikit.

2.3 *Software Defined Network* (SDN)

Software Defined Network (SDN) merupakan sebuah paradigma dalam mengelola, mendesain dan menerapkan jaringan. Paradigma ini hadir untuk mengatasi permasalahan pada jaringan semakin kompleks. Menurut organisasi dari pengembang *Software Defined Network* yaitu Open Networking Foundation, mendefinisikan bahwa *Software Defined Network* merupakan suatu arsitektur jaringan yang memisahkan sistem *control plane* dan *data plane.* Pada arsitektur jaringan secara konvensional, *control plane* dan *data plane* diimplementasikan secara langsung pada perangkat keras jaringan seperti router, *Switch* dan lain-lain (Nadeau & Gray, 2013). Namun, pada arsitektur *Software Defined Network*

sistem *control plane* dihilangkan dari perangkat keras jaringan dan diimplementasikan dalam bentuk software. Sehingga konfigurasi pada *control plane* akan bersifat *programmable* dan fleksibel. Berikut ini adalah gambaran umum arsitektur *Software Defined Network*.



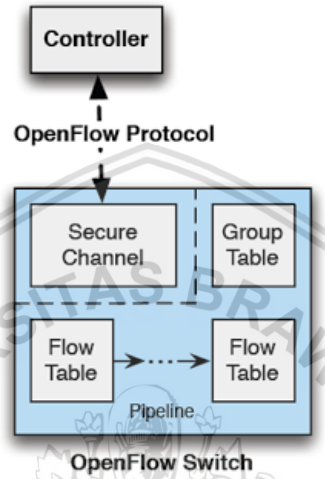
Gambar 2.3 Arsitektur *Software Defined Network*

Pada Gambar 2.3 terdapat 3 layer atau lapisan sistem dalam arsitektur *Software Defined Network* yakni:

1. *Application Plane,* merupakan layer yang berisi aplikasi-aplikasi yang terdapat dalam jaringan seperti aplikasi monitoring jaringan dan lain-lain. Layer ini dapat berkomunikasi dengan Layer *Control Plane* melalui *NorthBound Application Programming Interface*.
2. *Control Plane,* merupakan layer pertengahan yang bertanggungjawab untuk memberikan intruksi kepada *Data Plane* terhadap paket yang datang ke jaringan. Layer ini dapat berkomunikasi dengan Layer *Data Plane* dengan menggunakan protokol *OpenFlow.*
3. *Data Plane,* terdiri dari perangkat keras jaringan yang bertugas untuk mengatur trafik pada jaringan sesuai dengan intruksi dari *Control Plane*. Selain itu layer ini juga akan menyimpan informasi tentang intruksi dari *Control Plane*.

2.3.1 Open*Flow*

Open*Flow* adalah protokol komunikasi standar terbuka yang dapat memisahkan antara sistem *control plane* dan sistem *data plane* dari sebuah perangkat jaringan, selain itu juga dapat menciptakan komunikasi yang sangat baik antara *control plane* dan *data plane* (Open Networking Foundation, 2009). Open*Flow* dapat menghubungkan secara langsung interaksi antara SDN *Controller* dengan SDN *Switch* seperti yang terlihat pada Gambar 2.4.



Gambar 2.4 *Openflow Controller* dan *Openflow Switch*

2.3.2 SDN *Controller*

Controller merupakan salah satu komponen penting pada jaringan *Software Defined Network* (SDN). Komponen ini mampu mengelola dan mengontrol aliran data ke *Switch* menggunakan *Openflow Protocol* seperti yang terlihat pada Gambar 2.4. Selain itu, *Controller* juga dapat mengimplementasikan aplikasi jaringan seperti *routing*, *load balancing*, *intrusion detection* dan lain sebagainya. Pada saat ini banyak sekali *Controller* yang dapat digunakan pada arsitektur SDN baik yang komersil maupun *Open Source*.

Salah satu *Controller* pada arsitektur SDN yang dapat digunakan dalam penelitian ini adalah POX *Controller*. POX *Controller* merupakan perangkat lunak jaringan yang ditulis dengan bahasa pemrograman Python versi 2.7. POX *Controller* juga mendukung komunikasi antara *Control Plane* dan *Data Plane* dengan protokol Open*Flow* versi 1.0 (McCauley, 2015).

2.4 *Web Server*

Web Server merupakan suatu perangkat baik perangkat keras maupun lunak yang menggunakan protokol HTTP atau HTTPS sebagai media dalam mengakses berkas-berkas di dalam suatu halaman website dengan menggunakan web browser. Selain mengakses berkas web, penggunaan *Web Server* dapat sebagai penempatan suatu halaman web, sebagai penyimpanan atau database serta penggunaan aplikasi bisnis.

Berdasarkan penjelasan di atas dapat disimpulkan bahwa fungsi *Web Server* merupakan untuk mengirimkan atau mentransferkan data yang diminta oleh user yaitu sebuah halaman web dengan protokol HTTP atau HTTPS.

2.5 Mininet (SDN Network Simulator)

Mininet adalah sebuah alat yang dapat digunakan untuk mensimulasikan perangkat jaringan seperti host, *Switch*, router, *Controller* dan lain-lain dalam bentuk perangkat lunak (Lantz, 2015). Mininet juga mendukung pengembangan aplikasi jaringan dengan arsitektur jaringan SDN dan Open*Flow*. Dengan menggunakan perangkat lunak ini, memungkinkan kita untuk dapat mendesain infrastruktur jaringan melalui *Command Line Interface (CLI)* maupun *Graphical User Interface (GUI)*. Pada Mininet, terdapat beberapa komponen yang dapat mendukung simulasi jaringan antara lain:

1. Hosts, merupakan komponen jaringan yang dapat digunakan sebagai *server* ataupun *Client*. Komponen ini berjalan layaknya sebagai shell proses seperti *bash* pada linux sehingga dapat menjalankan program yang ada dalam sistem operasi dimana Mininet tersebut terpasang (Lantz, 2015).
2. *Switch*s, komponen ini dapat bertindak sebagai *Switch* atau *Data Plane* pada arsitektur SDN karena komponen ini dapat menggunakan Open*Flow* sebagai media komunikasi (Lantz, 2015).
3. Links, merupakan komponen yang dapat menghubungkan antara komponen satu dengan yang lainnya. Sebagai contoh antara Hosts dengan *Switch*, *Controller* dengan *Switch* dan lain sebagainya. Dalam penerapan pada perangkat fisik, Links dapat diasumsikan sebagai Ethernet atau Kabel yang menghubungkan 2 *interfaces* (Lantz, 2015).

2.6 Psutils

Psutils merupakan sebuah library yang berfungsi untuk mendapatkan informasi mengenai proses yang sedang berjalan pada sistem komputer. Selain itu library ini juga dapat memantau sumber daya sistem seperti *CPU*, *memory*, *disks*, *network* dan *sensors* (Psutil Documentation, 2018). Library ini dibangun dengan basis bahasa pemrograman *Python* dan dapat digunakan pada berbagai *platform* seperti *Linux, Windows, FreeBSD, OSX* dan lain-lain.

2.7 Apache Jmeter

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

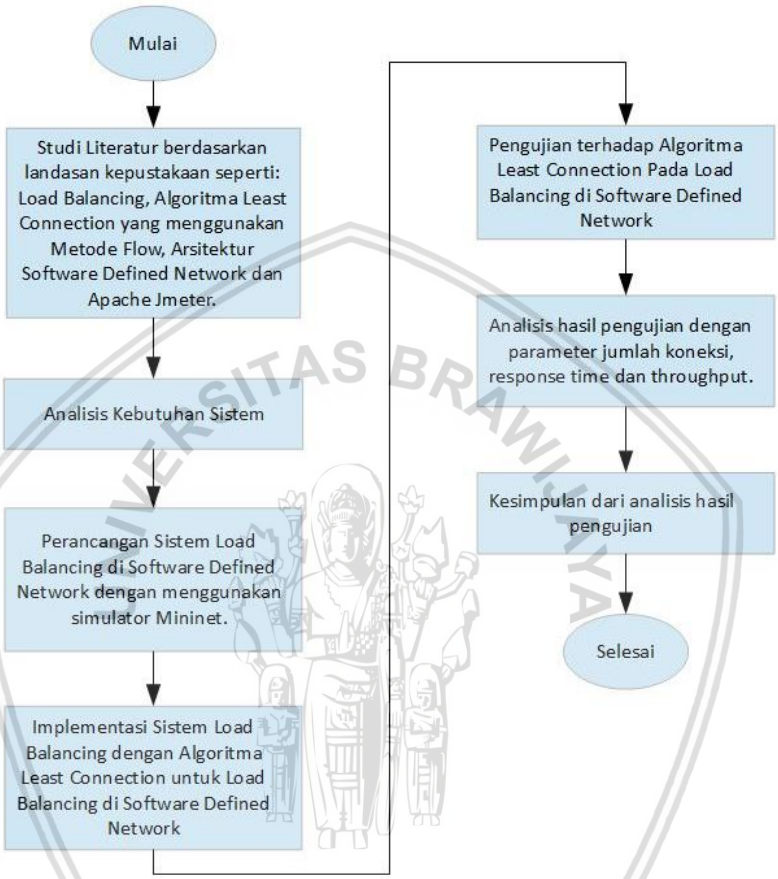
untuk menguji kinerja dari aplikasi, server atau protokol tertentu seperti Web (HTTP, HTTPS, ASP.NET, NodeJS), REST/SOAP Webservices, FTP dan lain-lain.

Untuk dapat menggunakan Apache Jmeter, adapun beberapa *properties* yang harus diketahui antara lain:

1. *Number of Thread (users)* merupakan jumlah pengguna yang dapat digunakan untuk simulasi *request* ke aplikasi yang akan diuji (Apache JMeter, 2018).
2. *Ramp-Up Period In Second* merupakan elemen yang menentukan berapa lama waktu (dalam detik) yang dibutuhkan untuk menjalankan *thread* yang telah ditentukan pada *Number of Thread (users)* (Apache JMeter, 2018). Jika menggunakan 5 *thread* dan *Ramp-Up Period In Second* adalah 100 detik, maka Apache Jmeter akan menjalankan seluruh thread selama 100 detik. Setiap thread akan dieksekusi setelah 20 (100/5) detik dari thread sebelumnya.
3. *Retrieve All Embedded Resources* merupakan elemen yang mengizinkan setiap *request* yang dilakukan oleh pengguna untuk mengakses resources pada halaman HTML seperti *Javascript, images, CSS dan lain-lain.*

BAB 3 METODOLOGI

Pada bagian ini akan dijelaskan tentang langkah yang dilakukan dalam penelitian. Berikut merupakan tahapan-tahapan metodologi penelitian yang digambarkan dengan diagram alir.



Gambar 3.1 Diagram Alir Metodologi Penelitian

Dalam gambar 3.1 tersebut menjelaskan metode penelitian yang akan di laksanakan oleh peneliti. Langkah pertama adalah mencari dasar teori atas penelitian baik melalui jurnal, buku maupun artikel yang berkaitan dengan penelitian. Lalu dilanjutkan dengan menganalisis kebutuhan dalam penelitian, seperti kebutuhan apa saja yang diperlukan dalam mengimplementasikan penelitian. Baik kebutuhan secara fungsional maupun kebutuhan non fungsional. Kemudian melakukan perancangan sebagai dasar dalam pengimplementasian sistem tersebut. Setelah itu, dilakukan pengujian dan analisis hasil untuk dilakukan penarikan kesimpulan terhadap penelitian.

3.1 Studi Literatur

Studi literatur merupakan tahap-tahapan mempelajari dasar-dasar teori yang akan digunakan sebagai referensi dalam penelitian. Tahapan ini juga

berfungsi untuk membantu penyelesaian masalah yang terdapat dalam penelitian agar tujuan dalam penelitian ini dapat tercapai. Adapun teori-teori yang dijadikan referensi dalam penelitian diperoleh dari buku, jurnal yang terkait dengan topik penelitian dan penelitian-penelitian sebelumnya. Studi literatur yang ada dalam penelitian ini meliputi *Load balancing, Software Defined Network, Web Server* dan *Apache Jmeter*.

3.2 Rekayasa Kebutuhan

Rekayasa kebutuhan merupakan sebuah proses yang dilakukan dalam penelitian yang bertujuan untuk mendapatkan informasi tentang kebutuhan sistem. Dalam hal rekayasa perangkat lunak, rekayasa kebutuhan merupakan bagian dari proses kebutuhan perangkat lunak yang berperan menjembatani jurang yang sering terjadi antara level rekayasa kebutuhan dan perancangan perangkat lunak (Pressman, 2002).

3.3 Perancangan dan Implementasi

3.3.1 Perancangan

Pada bagian ini, akan dipaparkan tentang bagaimana membuat perancangan terhadap sistem yang akan dibangun. Tahapan perancangan sistem adalah sebagai berikut:

1. Perancangan arsitektur *Software Defined Network* (SDN).
2. Perancangan sistem *load balancing*.
3. Perancangan algoritme *Least Connection*.
4. Perancangan algoritme pada *Agen Psutils*.
5. Perancangan *Client*.

3.3.2 Implementasi

Pada bagian ini, sistem akan diimplementasikan berdasarkan perancangan yang telah dibuat. Berikut ini implementasi sistem sebagai berikut:

1. Implementasi arsitektur *Software Defined Network* (SDN).
2. Implementasi sistem *load balancing*.
3. Implementasi algoritme *Least Connection*.
4. Implementasi algoritme pada *Agen Psutils*.
5. Implementasi *Client*.

3.4 Pengujian

Pada tahap ini, akan dilakukan pengujian terhadap sistem yang telah dibangun disertai analisis terhadap hasil dari pengujian tersebut. Pengujian

tersebut akan mengacu kepada beberapa parameter antara lain jumlah koneksi, response time dan throughput. Sehingga disusun beberapa rencana pengujian sebagai berikut:

1. Pengujian download untuk mengetahui kemampuan dari algoritme *Least Connection* yang menggunakan metode berbasis *flow* dan metode berbasis *Agen Psutils* ketika melayani *request* pada berkas di *server* dengan jumlah paket TCP yang banyak dan ukuran berkas yang besar
2. Pengujian kinerja dari sistem *load balancing* dengan parameter distribusi trafik, penggunaan sumber daya (CPU dan Memory) dari setiap *server* didalam cluster dan response time.

3.5 Kesimpulan

Setelah melalui tahap studi literatur, rekayasan kebutuhan, perancangan dan implementasi serta pengujian, maka sampailah pada tahap pengambilan kesimpulan. Pada bagian ini juga merupakan jawaban dari permasalahan yang terdapat pada rumusan masalah yang sudah ditulis. Selain itu juga akan dipaparkan beberapa saran terkait pengembangan penelitian ini kedepannya. Sehingga saran-saran tersebut dapat dipertimbangan pada penelitian-penelitian selanjutnya.

BAB 4 REKAYASA KEBUTUHAN

4.1 Deskripsi Umum Sistem

Sistem *load balancing* dengan menggunakan algoritme *Least Connection* ini merupakan sistem penyeimbang beban trafik untuk layanan web dengan jumlah koneksi yang terjalin sebagai faktor pembagian beban tersebut. Tidak seperti pada penelitian sebelumnya yang menggunakan *flow* sebagai penentu jumlah koneksi pada *server*, melainkan sistem ini menggunakan jumlah koneksi sebenarnya yang ada pada *server*. Untuk menerapkan sistem tersebut, penelitian ini akan menggunakan sebuah agen yang akan dipasangkan pada setiap *server* didalam *cluster* yang disebut dengan *Agen Psutils.* Agen tersebut berfungsi untuk mengirimkan jumlah koneksi dari *server* kepada *Controller*, yang mana informasi dari agen tersebut akan dikelola untuk algoritme *Least Connection* yang telah diterapkan pada *Controller* tersebut. Selain itu, sistem ini juga akan diterapkan pada arsitektur *Software Defined Network*. Sehingga dibutuhkan sebuah lingkungan kerja yang mendukung arsitektur tersebut. Untuk itu, penelitian ini akan menggunakan *Mininet* sebagai simulator.

Sistem *load balancing* ini diterapkan pada arsitektur *Software Defined Network,* sehingga terdapat subsistem yang saling bekerja sama untuk membentuk suatu sistem *load balancing* yang utuh. Subsistem tersebut yaitu Subsistem *Controller*, Subsistem *Switch*, Subsistem *Web Server*, Subsistem *Agen Psutils* dan Subsistem *Client*. Berikut ini penjelasan mengenai masing-masing subsistem:

- Subsistem *Controller* merupakan inti dari sebuah jaringan pada *Software Defined Network* yang bertugas untuk mengatur atau memutuskan aksi tertentu terhadap paket yang datang ke jaringan. Pada subsistem ini juga akan diterapkan metode *load balancing* menggunakan algoritme *Least Connection*.
- Subsistem *Switch* merupakan salah satu unit yang terdapat pada *Data Plane.* Subsistem ini yang berfungsi untuk menerima paket yang datang ke jaringan dan melakukan aksi terhadap paket tersebut berdasarkan informasi yang tersimpan didalam *flow table*. *Flow table* merupakan informasi mengenai paket yang masuk ke jaringan yang tersimpan didalam Subsistem *Switch*. Informasi tersebut akan digunakan untuk menentukan apa yang harus dilakukan terhadap paket tersebut.
- Subsistem *Web Server* adalah subsistem yang akan menyediakan layanan web pada sistem *load balancing* ini. Pada subsistem ini, terdapat beberapa *Web Server* yang secara bergantian melayani permintaan terhadap layanan. Setiap *Web Server* akan memiliki layanan yang sama (tersinkronisasi).

- Subsistem *Agen Psutils* merupakan subsistem yang akan mengumpulkan informasi mengenai jumlah koneksi yang terdapat pada *Web Server*. Kemudian informasi tersebut akan dikirimkan ke Subsistem *Controller*. Subsistem ini tertanam pada setiap *Web Server*, sehingga Subsistem *Controller* akan memiliki informasi jumlah koneksi dari semua *Web Server*.
- Subsistem *Client* merupakan unit opsional atau tambahan pada sistem. Subsistem ini hanya akan digunakan pada saat pengujian sistem *load balancing*. Subsistem ini bertugas untuk mengirimkan *request* terhadap layanan web yang ada pada sistem.

4.2 Deskripsi Kebutuhan Sistem

Untuk mempermudah proses perancangan dan implementasi sistem *load balancing*, penelitian ini akan memaparkan beberapa kebutuhan sistem yaitu kebutuhan fungsional, kebutuhan non-fungsional dan kebutuhan lingkungan kerja.

4.2.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan kebutuhan yang menunjang sistem agar dapat berjalan sesuai dengan kemampuan yang telah didefinisikan pada deskripsi umum sistem. Berikut ini merupakan kebutuhan fungsional sistem, yaitu:

4.2.1.1 Subsistem *Controller*

1. Subsistem harus dapat menerima paket dari Subsistem *Switch*.
2. Subsistem harus dapat menerima paket atau informasi mengenai jumlah koneksi pada *Web Server* yang dikirimkan oleh Subsistem *Agen Psutils*.
3. Subsistem harus dapat menentukan atau mengarahkan paket ke *Web Server* dengan jumlah koneksi paling sedikit.
4. Subsistem harus dapat menjalankan modul *load balancing* menggunakan POX versi carp.

4.2.1.2 Subsistem *Switch*

1. Subsistem harus dapat menerima paket dari Subsistem *Client*.
2. Subsistem harus dapat mengirimkan pesan informasi paket yang datang kepada Subsistem *Controller* (*Packet In Message*).

3. Subsistem harus dapat menerima pesan informasi mengenai tindakan yang harus dilakukan terhadap paket, dimana pesan tersebut dikirimkan oleh Subsistem *Controller* (*Packet Out Message*).
4. Subsistem harus dapat berjalan pada perangkat lunak *Open vSwitch* versi 2.8.1.

4.2.1.3 Subsistem *Web Server*

1. Subsistem harus dapat menyediakan layanan website.
2. Subsistem harus dapat memberikan *response* terhadap *request* dari Subsistem *Client*.

4.2.1.4 Subsistem *Agen Psutils*

1. Subsistem harus dapat mengetahui jumlah koneksi pada Subsistem *Web Server*.
2. Subsistem harus dapat mengirimkan informasi mengenai jumlah koneksi pada Subsistem *Web Server* kepada Subsistem *Controller*.

4.2.1.5 Subsistem *Client*

1. Subsistem harus dapat mengirimkan *request* terhadap layanan website pada Subsistem *Web Server*.

4.2.2 Kebutuhan Non-Fungsional

Kebutuhan non-fungsional merupakan kebutuhan terhadap batasan layanan yang ditawarkan oleh sistem. Kebutuhan non-fungsional sistem tersebut adalah sebagai berikut:

1. Sistem hanya diterapkan pada sistem operasi Linux Ubuntu 16.04.
2. Sistem hanya diterapkan dengan menggunakan 3 *server* didalam cluster yang memiliki layanan website yang sama (tersinkronisasi).
3. Sistem hanya diterapkan pada jaringan lokal.
4. Sistem tidak menggunakan domain sebagai identifikasi alamat IP *Web Server* atau alamat IP publik melainkan hanya menggunakan alamat IP virtual.

4.2.3 Kebutuhan Lingkungan Kerja Sistem

Pada bagian ini akan dideskripsikan kebutuhan lingkungan kerja peneliti untuk menerapkan sistem load balancing. Kebutuhan tersebut meliputi kebutuhan perangkat lunak dan kebutuhan perangkat keras yang menunjang proses kerja dari sistem.

4.2.3.1 Kebutuhan Perangkat Lunak

Berikut ini adalah perangkat lunak yang digunakan pada lingkungan kerja peneliti antara lain:

1. Sistem operasi *Linux Ubuntu* 16.04.
2. Simulator *Mininet* versi 2.1.0.
3. *Open vSwitch* versi 2.8.1.
4. *Apache Jmeter* sebagai perangkat lunak yang digunakan untuk menguji kinerja dari sistem *load balancing*.

4.2.3.2 Kebutuhan Perangkat Keras

Kebutuhan perangkat keras pada sistem ini yaitu sebuah Komputer atau Laptop dengan spesifikasi sebagai berikut:

1. Hawlett Packard Notebook 14 Series.
2. Processor Intel Celeron Bay Trail-M @ 2,16 GHz.
3. RAM 4 GB.
4. Hardisk 320 GB SATA 7200 RPM.

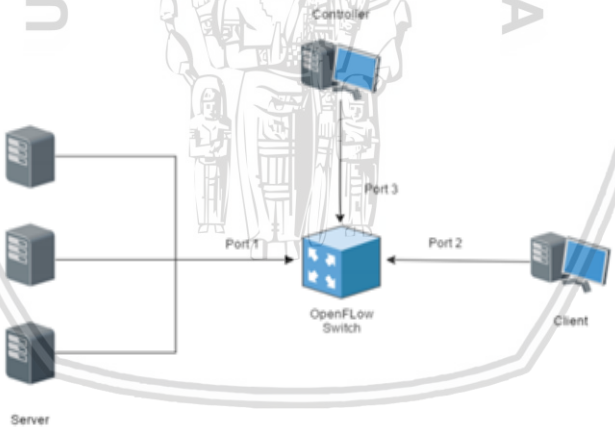
BAB 5 PERANCANGAN DAN IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai perancangan dan implementasi sistem. Selain itu juga akan dijelaskan lebih lengkap mengenai perancangan dan implementasi sistem *Load balancing*, Arsitektur SDN, Algoritme *Least Connection*, Algoritme Pada *Agen Psutils* dan *Client*.

5.1 Perancangan

5.1.1 Perancangan Arsitektur SDN

Untuk membangun sebuah arsitektur jaringan *Software Defined Network* (SDN) dibutuhkan sedikitnya 1 buah *Switch* yang mendukung komunikasi dengan protokol Open*Flow* dan sebuah *Controller*. Pada bagian rekayasa kebutuhan lebih tepatnya pada deskripsi kebutuhan fungsional sistem, telah didefinisikan kebutuhan pada sistem antara lain Subsistem *Controller*, Subsistem *Switch*, Subsistem *Web Server*, Subsistem *Agen Psutils* dan Subsistem *Client*. Penelitian ini akan menggunakan Mininet untuk membangun sistem *load balancing* pada arsitektur SDN berdasarkan kebutuhan tersebut. Gambar 5.1 merupakan gambaran tentang arsitektur *load balancing* pada *Software Defined Network* yang akan dibangun.



Gambar 5.1 Arsitektur *Software Defined Network*

5.1.1.1 POX *Controller*

POX merupakan sebuah library yang ditulis dengan bahasa pemrograman Python versi 2.7. POX juga mendukung komunikasi antara *Control Plane* dan *Data Plane* dengan protokol Open*Flow* versi 1.0. Pada penelitian ini, sistem yang melakukan penyeimbang beban (*load balancer*) akan dibuat menggunakan library POX. Load balancer ini akan menginstruksikan *Switch* untuk meneruskan paket dan pada saat yang bersamaan juga akan menyeimbangkan beban trafik menggunakan algoritme *Least Connection* dan *Agen Psutils*.

5.1.1.2 *Open vSwitch*

Open vSwitch adalah aplikasi yang bertindak sebagai Open*Flow Switch* pada arsitektur SDN. Selain itu, aplikasi ini juga dapat digunakan sebagai virtual *Switch* pada Mininet simulator.

5.1.1.3 *Web Server*

Pada penelitian ini, *Web Server* yang akan digunakan adalah modul *Web Server* pada Python yaitu SimpleHTTP*Server*. Modul tersebut secara langsung sudah terdapat pada modul standar Python.

5.1.1.4 Alokasi Pengalamatan IP

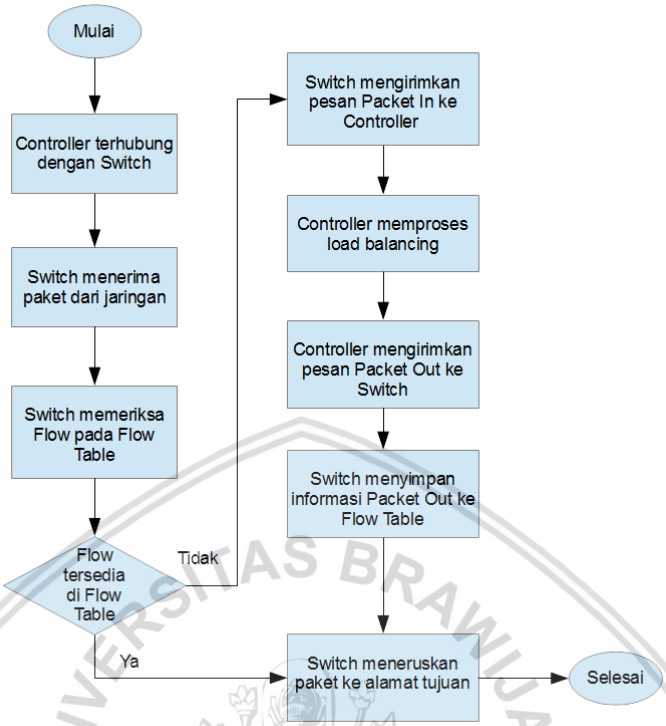
Berikut ini alokasi pengalamatan IP untuk setiap *server* sebagai berikut:

Komponen	Alamat IP	MAC
Web Server 1	10.0.0.1	00:00:00:00:00:01
Web Server 2	10.0.0.2	00:00:00:00:00:02
Web Server 3	10.0.0.3	00:00:00:00:00:03
Client	10.0.0.4	00:00:00:00:00:04

5.1.2 Perancangan Sistem *Load balancing*

5.1.2.1 Alur Sistem Secara Umum

Proses pendistribusian beban secara seimbang pada sistem *load balancing* diawali dengan datangnya trafik dari jaringan ke sistem. Kemudian trafik tersebut akan diteruskan oleh *Switch* ke *Controller*. Selanjutnya *Controller* yang akan menentukan dan memberikan instruksi kepada *Switch* untuk aksi yang harus dilakukan selanjutnya. Pada *Controller* juga akan diekseskusi mekanisme pembagian beban trafik ke *server* berdasarkan algoritme *Least Connection* atau *server* yang memiliki koneksi paling sedikit. Untuk lebih jelas dapat dilihat pada Gambar 5.2.



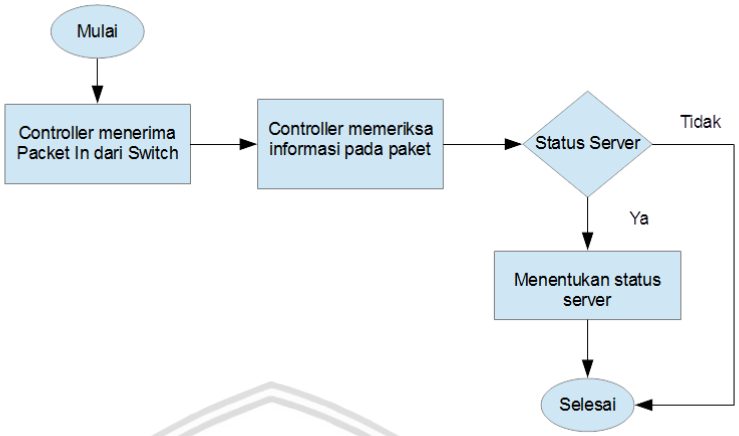
Gambar 5.2 Alur Sistem Secara Umum

Dari Gambar 5.2 kita dapat melihat bahwa alur sistem *load balancing* diawali dengan terhubungnya *Switch* ke *Controller*. Kemudian *Switch* akan menerima paket yang masuk ke sistem. Sebelum mengirimkan Paket In ke *Controller*, terlebih dahulu *Switch* memeriksa entri *flow* pada *Flow* table. Jika entri *flow* tidak tersedia maka *Switch* akan mengirimkan pesan Packet In ke *Controller* dan *Controller* akan memproses *load balancing* dengan menentukan alamat tujuan dari paket tersebut berdasarkan *server* dengan jumlah koneksi paling sedikit. Setelah *Switch* menerima pesan Packet Out dari *Controller*, *Switch* akan menyimpan entri ke *Flow* table. Sehingga jika terdapat paket dengan informasi yang sama, *Switch* tidak mengirimkan pesan Packet In ke *Controller*. Setelah itu paket akan diteruskan berdasarkan alamat tujuan dari paket tersebut.

5.1.2.2 Alur Sistem *Load balancing*

Alur sistem *load balancing* ini merupakan mekanisme pembagian beban yang terdapat pada *Controller*. Pembagian beban dilakukan berdasarkan algoritme *Least Connection* yang dimana algoritme tersebut akan menentukan arah trafik kepada *server* dengan jumlah koneksi paling sedikit. Didalam mekanisme pembagian beban terdapat alur pemeriksaan status *server*, pemeriksaan paket dari *Client* dan pemeriksaan paket dari *server*.

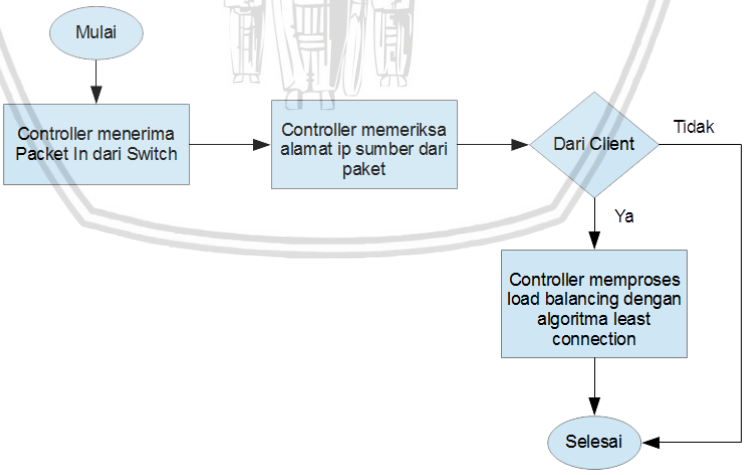
a. Alur Pemeriksaan Status *Server*



Gambar 5.3 Alur Pemeriksaan Status *Server*

Gambar 5.3 merupakan tahap-tahap pemeriksaan informasi status dari *server* dengan memanfaatkan Packet In pada *Controller*. Paket ini akan dikirimkan oleh setiap *server* dalam jangka waktu tertentu menggunakan protokol ARP. Jika *server* yang terdaftar didalam cluster mengirimkan paket dengan jenis ini, maka *server* tersebut dianggap sedang beroperasi atau sedang *up.* Namun, jika *server* tidak mengirimkannya maka *server* akan dianggap sedang *down*.

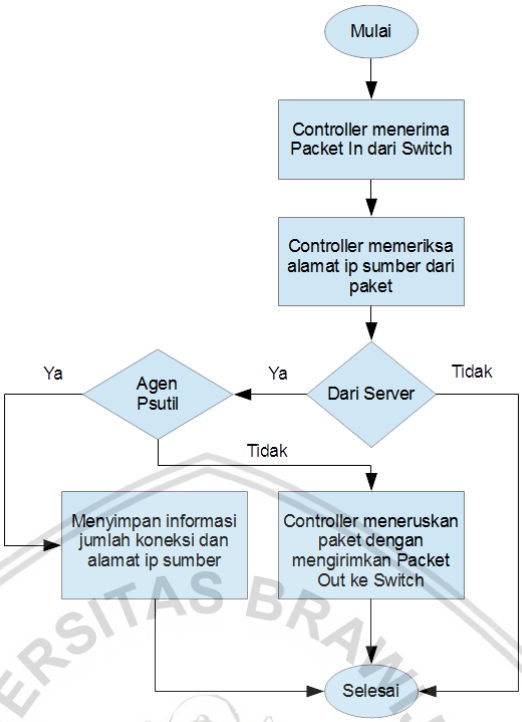
b. Alur Pemeriksaan Paket dari *Client*



Gambar 5.4 Alur Pemeriksaan Paket dari *Client*

Gambar 5.4 merupakan proses *load balancing* yang dilakukan jika paket tersebut datang dari *Client* dengan memeriksa alamat ip sumber paket dan alamat ip tujuan dari paket tersebut.

c. Alur Pemeriksaan Paket dari *Server*

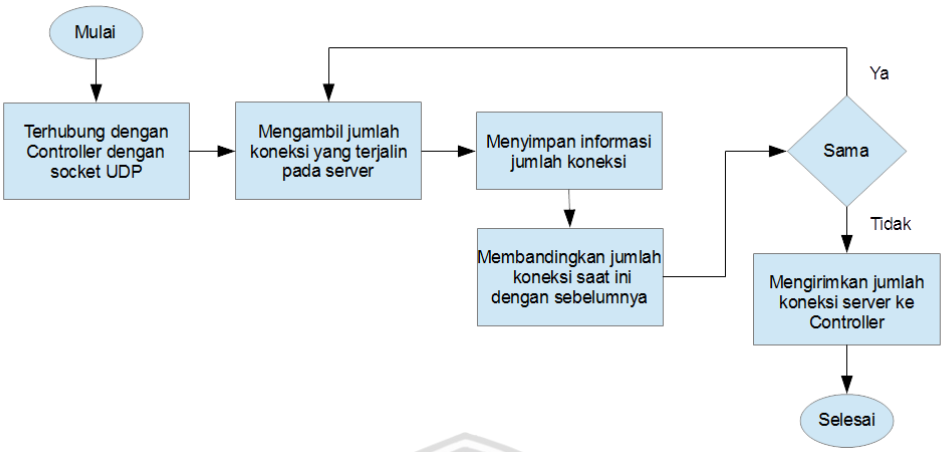


Gambar 5.5 Alur Pemeriksaan Paket dari *Server*

Pada bagian ini terdapat beberapa jenis paket yang dikirimkan oleh *server* yaitu paket dengan ip tujuan ke *Client* dan paket dari *Agen Psutils* seperti yang terlihat pada Gambar 5.5. Jika paket tersebut merupakan paket dari *Agen Psutils* pada *server* maka *Controller* akan menyimpan informasi dari *Agen Psutils*. Dimana paket tersebut merupakan informasi mengenai jumlah koneksi dan alamat ip sumber. Namun jika bukan dari *Agen Psutils* maka *Controller* akan meneruskan paket dengan mengirimkan Paket Out ke *Switch* untuk diteruskan ke alamat tujuan dari paket.

5.1.2.3 Alur Pengiriman Jumlah Koneksi

Alur pengiriman jumlah koneksi merupakan alur yang terdapat pada *Agen Psutils*. Agen ini akan terpasang pada setiap *server* dan mengirimkan jumlah koneksi pada *server* secara *real time*. Berikut ini gambaran alur pengiriman jumlah koneksi.



Gambar 5.6 Alur Pengiriman Jumlah Koneksi

Berdasarkan pada Gambar 5.5, pengiriman jumlah koneksi oleh *Agen Psutils* diawali dengan terhubungnya *server* dengan *Controller* menggunakan socket UDP. Kemudian akan dilakukan pengambilan jumlah koneksi pada *server* dan menyimpan informasi tersebut. Selanjutnya koneksi sebelumnya akan dibandingkan dengan koneksi saat ini. Jika koneksi saat ini dengan sebelumnya berbeda maka jumlah koneksi saat ini akan dikirimkan ke *Controller*. Mekanisme ini akan mengurangi beban jaringan dengan tidak mengirimkan informasi yang sama secara terus menerus.

5.1.3 Perancangan Algoritme *Least Connection*

Algoritme *Least Connection* akan terpasang pada modul *load balancing* yang akan dijalankan menggunakan POX *Controller*. Tabel 5.1 merupakan pseudocode dari algoritme *Least Connections*.

Tabel 5.1 Pseudocode Algoritme *Least Connection*

Kode Sumber: Algoritme Least Connection	
1	Begin
2	Receive connection information from Agen Psutils.
3	Store connection information to **webServer**'s variable.
4	Set **minimumConnection** is *Web Server* 1.
5	If **webServer** is None:
6	Return **minimumConnection**
7	For **webConnection** in **webServer**:
8	If **webConnection** is less then **minimumConnection**:
9	**minumumConnection** is equal to **webConnection**
10	Endif
11	Endfor

```
12      Return minimumConnection
13  End
```

Berikut ini penjelasan dari Tabel 5.1 sebagai berikut:

1. Pada baris ke 2, menerima informasi jumlah koneksi dari *Agen Psutils*.
2. Pada baris ke 3, menyimpan informasi yang diterima dari *Agen Psutils* pada sebuah variabel.
3. Pada baris ke 4, penentuan awal *server* dengan jumlah koneksi paling sedikit yaitu *Web Server* 1.
4. Pada baris ke 5 hingga 6 merupakan pemeriksaan informasi mengenai jumlah koneksi. Jika informasi dari *Agen Psutils* belum ada atau belum diterima maka akan dikembalikan nilai dari *minimumConnection*.
5. Pada baris ke 7 hingga 10 merupakan perulangan dan perbandingan jumlah koneksi antara satu *Web Server* dengan *Web Server* lainnya.
6. Pada baris ke 12 merupakan pengembalian nilai dari *minimumConnection*.

5.1.4 Perancangan Algoritme Pada *Agen Psutils*

Algorima ini merupakan algoritme yang terdapat pada *Agen Psutils*. Dimana merupakan tahap-tahap mendapatkan informasi jumlah koneksi hingga tahap-tahap mengirimkan informasi tersebut ke *Controller*. Tabel 5.2 merupakan pseudocode dari algoritme pada *Agen Psutils*.

Tabel 5.2 Pseudocode Algoritme pada *Agen Psutils*

```
Kode Sumber: Algoritme pada Agen Psutils
1   Begin
2       Get connection information from system using Psutil
3       Module.
4       Store connection information to totalConnection's
5       variable.
6       Connect to Controller using UDP socket.
7       While True:
8           If totalConnection's value is different from before:
9               Send totalConnection's value to Controller
10          Endif
11          If SystemExit is True:
12              Break
```

13	Endif
14	Endwhile
15	End

Berikut ini penjelasan mengenai pseudocode pada Tabel 5.2 sebagai berikut:

1. Pada baris ke 2, mengambil jumlah koneksi yang terjalin pada *server* dengan menggunakan modul Psutils.
2. Pada baris ke 3, menyimpan informasi yang didapat pada baris ke-1 pada variabel *totalConnection*.
3. Pada baris ke 4, menghubungkan *server* dengan *Controller* untuk mengirimkan informasi tersebut dengan menggunakan socket UDP.
4. Pada baris ke 5 hingga ke 8, perulangan dan pemeriksaan jumlah koneksi saat ini dan sebelumnya. Kemudian mengirimkan informasi tersebut ke *Controller*.
5. Pada baris ke 9 hingga ke 11, perulangan akan dihentikan ketika program memutuskan untuk menghentikan proses.

5.1.5 Perancangan *Client*

Client akan digunakan pada pengujian untuk mengirimkan *request* ke *Web Server*. Sehingga pada *Client* dibutuhkan aplikasi pengiriman *request* dengan parameter tertentu yaitu *Apache Jmeter*.

5.2 Implementasi

5.2.1 Implementasi Arsitektur SDN

5.2.1.1 Instalasi dan Konfigurasi Mininet

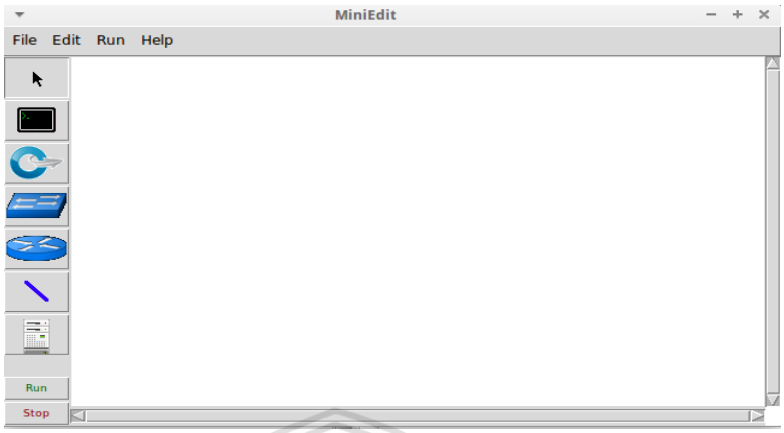
Mininet merupakan simulator jaringan yang dapat menerapkan arsitektur SDN secara virtual. Untuk menggunakan simulator tersebut kita dapat memasangkan simulator ini pada Linux dengan perintah:

```
$ sudo apt-get install mininet
```

Setelah melakukan instalasi mininet, topologi yang telah dirancang pada tahap perancangan arsitektur SDN dapat diterapkan pada Mininet Simulator dengan menjalankan aplikasi Mininet GUI (Miniedit) dengan menggunakan perintah sebagai berikut:

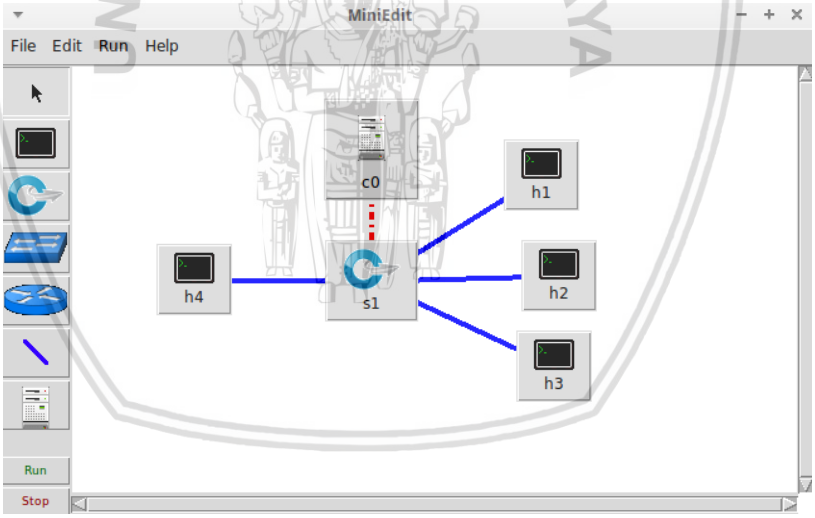
```
$ sudo python /opt/miniedit.py
```

Gambar 5.7 merupakan tampilan ketika Mininet GUI (Miniedit) dijalankan:



Gambar 5.7 Mininet GUI (Miniedit)

Kemudian dapat mengimplementasikan topologi dengan menambahkan 1 unit *Controller*, 1 unit *Switch* dan 4 unit host yang mana 1 unit host bertindak sebagai *Client* dan 3 lainnya bertindak sebagai *server*. Kemudian masing-masing unit dihubungkan oleh link dengan pengaturan bandwidth 80 Mbit. Dari rancangan tersebut didapatlah hasil seperti pada Gambar 5.8.



Gambar 5.8 Topologi Pada Mininet GUI (Miniedit)

5.2.1.2 Instalasi POX *Controller*

POX menyediakan banyak sekali modul yang dapat digunakan untuk berkomunikasi dengan *Switch* SDN menggunakan protokol Open*Flow*. Berikut adalah perintah yang digunakan untuk memasang dan menjalankan POX *Controller*.

```
$ sudo apt-get install git
$ git clone https://github.com/noxrepo/pox
$ cd pox/pox
```

Untuk memastikan POX *Controller* berjalan dengan baik adalah dengan menggunakan perintah sebagai berikut:

```
$ python pox.py forwarding.l2_learning
```

5.2.1.3 Instalasi *Open vSwitch*

Untuk menjalankan *Switch* SDN secara virtual dibutuhkan sebuah software yang mampu bertindak layaknya sebuah *Switch* fisik. Untuk itu kita menggunakan *Open vSwitch* dengan memasang pada lingkungan kerja sistem. Berikut tahap-tahap instalasi *Open vSwitch*:

```
$ sudo apt-get install openv-Switch-Switch
```

Dengan menjalankan perintah tersebut pada sistem operasi Linux maka *Switch* SDN telah terpasang.

5.2.1.4 Instalasi *Web Server*

Untuk dapat menjalankan *Web Server* dibutuhkan modul python yang bernama Flask. Namun sebelum dapat menginstall flask dibutuhkan python-pip yang merupakan package manager yang dimiliki oleh python. Perintah yang dilakukan adalah seperti berikut.

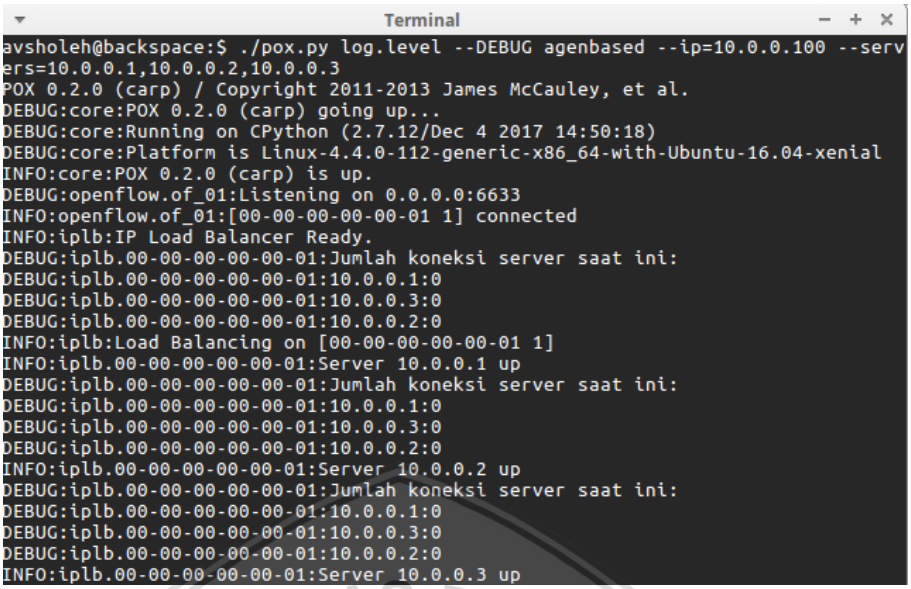
```
$ sudo apt-get install python-pip
$ pip install Flask
```

5.2.2 Implementasi Sistem *Load balancing*

Untuk melakukan implementasi loadbalancing di web*server* pada *Software Defined Network*, hal pertama yang dilakukan adalah melakukan inisialisasi topologi yang telah dideskripsikan pada tahap perancangan arsitektur SDN dan menjalankan Mininet GUI (Miniedit). Kemudian pada terminal, kita dapat menjalankan POX *Controller* dengan perintah sebagai berikut:

```
$ sudo python pox.py log.level --DEBUG agenbased --ip=10.0.0.100 --
servers=10.0.0.1,10.0.0.2,10.0.0.3
```

Jika proses berlangsung dengan baik maka akan terlihat seperti pada gambar.



Gambar 5.9 Tampilan Ketika POX *Controller* Dijalankan

Pada perintah seperti Gambar 5.9 dapat dijelaskan bahwa algoritme yang digunakan adalah agenbased atau *Agen Psutils*. Kemudian pada opsi *–ip=10.0.0.100* merupakan alamat IP yang digunakan oleh *Client* untuk mengakses *server*. Sedangkan pada opsi *–servers=10.0.0.1,10.0.0.2,10.0.0.3* merupakan daftar alamat IP dari host yang akan dijadikan sebagai *server*. Terlihat pada Gambar 5.9, bahwa *Controller* memberikan informasi bahwa alamat IP yang dijadikan sebagai *server* dalam kondisi hidup atau dapat melayani permintaan dari *Client*. Selain itu juga terdapat informasi jumlah koneksi dari masing-masing *server*.

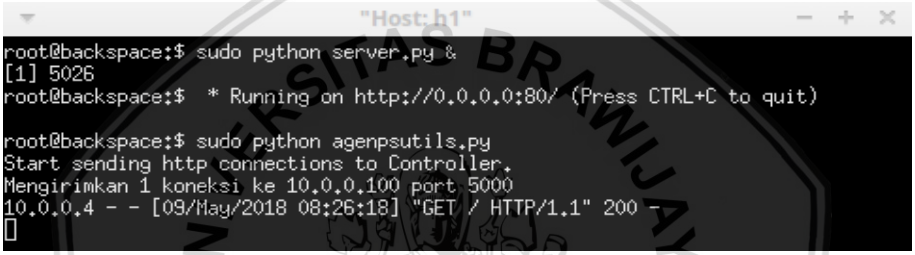
Kemudian, pada masing-masing *server* dapat dijalankan layanan web dan script agenpsutils dengan perintah sebagai berikut.

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

Jika proses berjalan dengan lancar, maka akan terlihat seperti pada gambar.

Gambar 5.10 Tampilan Terminal Pada *Server h1*

Pada Gambar 5.10 terlihat bahwa *server* dengan alamat IP 10.0.0.1 atau dengan alias h1 telah menjalankan layanan HTTP pada alamat IP publik yaitu 0.0.0.0 dan port 80 serta script agenpsutils sedang mengirimkan jumlah koneksi yang dimiliki saat ini. Jika terdapat *request* dari *Client* maka akan terlihat seperti pada gambar 5.11.



Gambar 5.11 Tampilan Terminal Pada *Server h1* Ketika Mendapatkan *Request*

5.2.3 Implementasi Algoritme *Least Connection*

Berikut ini penerapan pseudocode pada tahap perancangan algoritme *Least Connection* dalam bentuk kode sumber:

Tabel 5.3 Kode sumber Algoritme *Least Connection*

Kode Sumber: Algoritme Least Connection

```
1   /** kode sebelumnya **/
2   if len(self.total_connection) == 0:
3       return self.live_servers.keys()[0]
4   webServerIp = self.total_connection.keys()[0]
5   webServerConnection = self.total_connection[webServerIp]
6   for serverIp in self.total_connection:
7       if self.total_connection[serverIp] < webServerConnection:
8           webServerIp = serverIp
9           webServerConnection = self.total_connection[serverIp]
10  return webServerIp
11  /** kode setelahnya **/
```

Kode sumber diatas diterapkan pada POX *Controller* yang menggunakan Python sebagai bahasa pemrograman. Untuk menjalankan program tersebut kita dapat menggunakan perintah seperti dibawah ini:

```
$ sudo python pox.py log.level --DEBUG lbagent --ip=10.0.0.100 --servers=10.0.0.1,10.0.0.2,10.0.0.3
```

Dari perintah diatas dapat dijelaskan bahwa *pox.py* merupakan script utama pada POX *Controller* untuk menjalankan script yang telah dibuat yaitu *lbagent.py*. Kemudian terdapat parameter *log.level* yaitu opsi untuk menampilkan pesan-pesan pada mode *debug*. Selanjutnya diikuti dengan nama script *lbagent* sebagai script yang akan digunakan untuk *load balancing*. Kemudian parameter --ip=10.0.0.100 merupakan alamat IP virtual yang akan digunakan oleh *Client* untuk mengakses *Web Server* dan --*servers*=10.0.0.1,10.0.0.2,10.0.0.3 merupakan alamat IP dari *Web Server*.

5.2.4 Implementasi Algoritme Pada *Agen Psutils*

5.2.4.1 Instalasi Library Psutils

Psutil adalah modul python yang digunakan untuk mengambil data jumlah koneksi pada *server*. Modul tersebut dibutuhkan oleh *Agen Psutils* untuk mendapatkan data mengenai jumlah koneksi pada *server* dan mengirimkannya ke *Controller*. Perintah yang digunakan untuk menginstall psutil adalah sebagai berikut.

```
$ pip install psutil
```

5.2.4.2 Penerapan Algoritme Pada *Agen Psutils*

Agen Psutils diimplementasikan menggunakan bahasa pemrograman *Python* dengan memanfaatkan *socket* UDP untuk mengirimkan informasi yang didapat dari *Psutils.* Informasi yang dikirimkan berupa jumlah koneksi yang dimiliki oleh *server.*

Tabel 5.4 Kode sumber *Agen Psutils*

Kode Sumber: Agen Psutils	
1	`import socket`
2	`import psutil`
3	`import pickle`
4	`import time`
5	
6	`IP = '10.0.0.100'`
7	`PORT = 5000`

```
8
9   # Create UDP socket
10  server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
11  server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
12
13  # First instance total connection is 0
14  total_connections = 0
15
16  # Collect only http connections
17  def get_http_connections():
18      container = 0
19      for item in psutil.net_connections(kind='inet'):
20          if item[3][1] == 80 and item[3][0] != '0.0.0.0':
21              container += 1
22          return container
23
24  if __name__ == '__main__':
25      print "Start sending http connections to Controller."
26      # Iterating process of sending total connections
27      while True:
28          try:
29              if total_connections !=
30  len(get_http_connections()):
31                  server.connect((IP, PORT))
32                  total_connections =
33  len(get_http_connections())
34                  print 'Mengirimkan %s koneksi ke %s port %s'
35  % (total_connections, IP, PORT)
36                  server.send(pickle.dumps(total_connections))
37          # Will stop if Interruption keyboard is detected
            except KeyboardInterrupt:
                break
            time.sleep(2)
```

Pada Tabel 5.4 merupakan kode sumber dari program *Agen Psutils.* Berikut ini penjelasan dari *code* tersebut:

- Pada baris ke 1 sampai 4 merupakan *import* modul yang akan digunakan oleh program.

- Pada baris ke 6 dan 7 merupakan alamat IP dan *port* dari penerima informasi yakni *Controller.*
- Pada baris ke 10 dan 11 adalah proses pembuatan *socket UDP* dan menggunakan opsi *socket.SO_REUSEADDR* agar program dapat menggunakan alamat IP dan *port* yang sama kembali jika terjadi koneksi ulang terhadap *server*.
- Pada baris ke 14 merupakan *variable* tempat menyimpan jumlah koneksi yang dimiliki oleh *server.*
- Pada baris ke 18 sampai 23 adalah *method* yang menggunakan modul *Psutils* untuk mendapatkan informasi jumlah koneksi yang terjalin pada *server.*
- Pada baris ke 28 sampai 37 merupakan perulangan yang akan digunakan untuk mengirimkan informasi yang telah didapat ke *Controller* menggunakan *socket* UDP.

5.2.5 Implementasi *Client*

Client menggunakan aplikasi *Apache Jmeter* untuk mengirimkan *request* kepada *server* yang digunakan untuk pengujian. Perintah yang dilakukan untuk menginstall adalah sebagai berikut.

```
$ sudo apt-get install jmeter
```

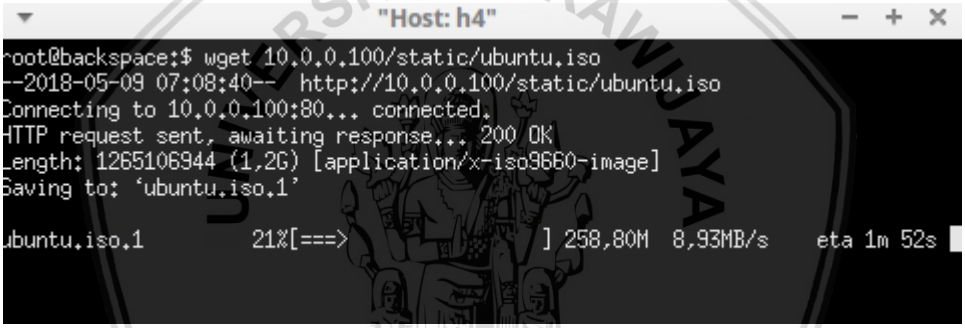
BAB 6 PENGUJIAN

6.1 Pengujian *Download*

Pengujian download ini bertujuan untuk mengetahui kemampuan dari algoritme *Least Connection* yang menggunakan metode berbasis *flow* dan metode berbasis *Agen Psutils* ketika melayani *request* pada berkas di *server* dengan jumlah paket TCP yang banyak dan ukuran berkas yang besar. Pada pengujian ini *Client* dengan alamat IP 10.0.0.4 akan melakukan *request* berkas pada *server* berukuran 1,2 GigaBytes dan bandwidth pada jaringan adalah 10 MegaBytes.

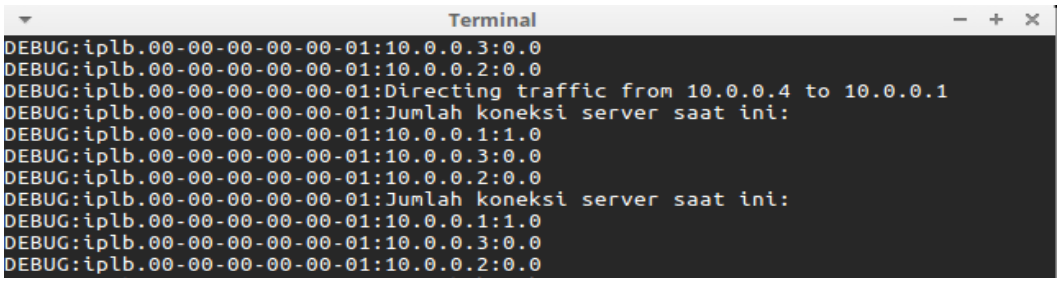
6.1.1 Metode Berbasis *Agen Psutils*

Berdasarkan skenario yang telah ditentukan, berikut ini merupakan hasil pengujian download pada metode berbasis *Agen Psutils*.



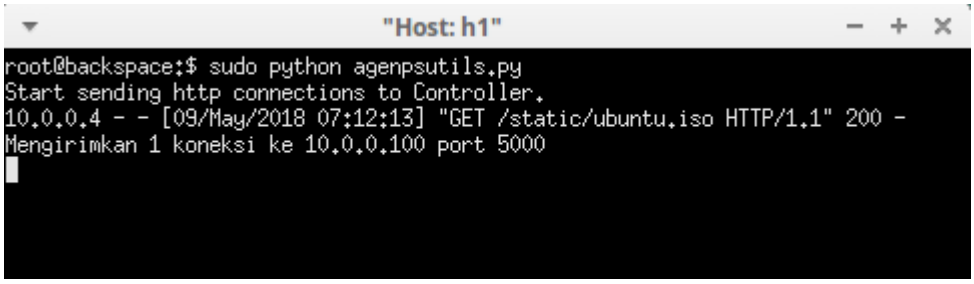
Gambar 6.1 *Client* Mengakses Data Pada *Server*

Pada Gambar 6.1 terlihat bahwa host h4 yang merupakan *Client*, mengakses alamat url *http://10.0.0.100/static/ubuntu.iso* dengan menggunakan wget untuk mengunduh berkas pada *server*. Kemudian *server* menanggapi permintaan dari *Client* dan mengirimkan berkas berukuran 1,2 GigaByte ke *Client*. Proses tersebut berlangsung dengan estimasi waktu 1 menit 52 detik.



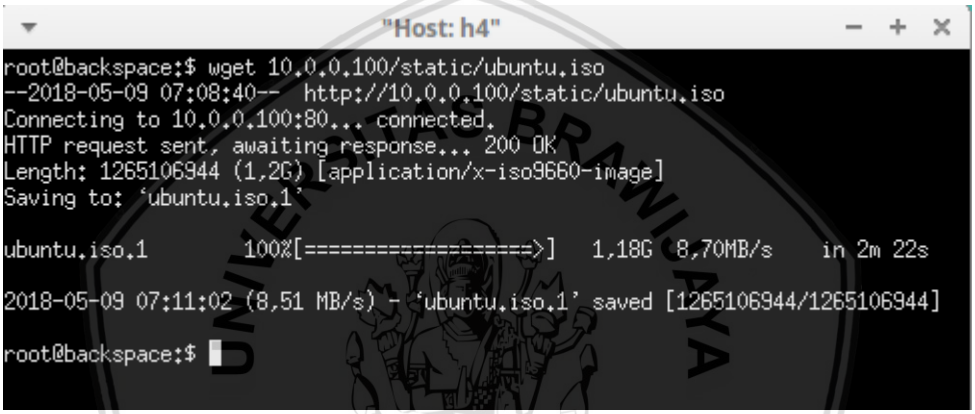
Gambar 6.2 *Controller* Mengarahkan Trafik ke *Server h1*

Pada sisi *Controller*, terlihat bahwa trafik yang datang diarahkan ke *server* dengan alamat IP 10.0.0.1. Selain itu juga, *Controller* menginformasikan jumlah koneksi dari setiap *server* secara berkala.



Gambar 6.3 *Server h1* Melayani *Request* dari *Client*

Pada *server* dengan IP 10.0.0.1 atau dengan alias H1 berhasil menerima permintaan dari *Client* dan menanggapi permintaan tersebut. Kemudian mengirimkan jumlah koneksi yang terhubung pada *server* tersebut ke *Controller*.

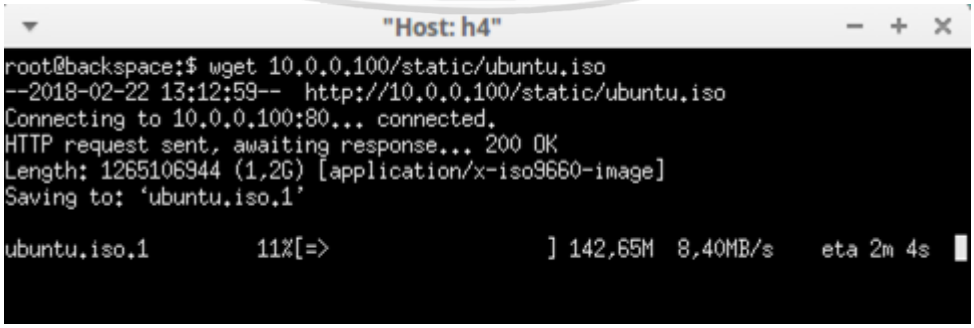


Gambar 6.4 *Request* data dari sisi *Client* Telah Selesai

Kemudian, kembali ke sisi *Client* dapat menerima unduhan berkas secara utuh dari *server* tanpa ada kendala dalam proses pengirimannya.

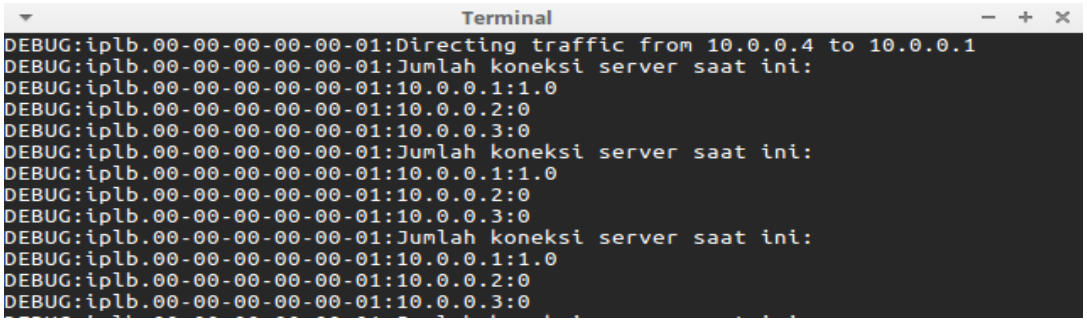
6.1.2 Metode Berbasis *Flow*

Dari skenario yang telah ditetapkan, didapatlah hasil pengujian pada metode berbasis *flow* sebagai berikut:



Gambar 6.5 *Client* Mengakses Data Pada *Server*

Pada sisi *Client* yang mencoba melakukan *request* dengan menggunakan tool wget pada url *http://10.0.0.100/static/ubuntu.iso*, *Client* berhasil menerima response dari *server* dan melakukan proses unduh berkas seperti telihat pada Gambar 6.5.

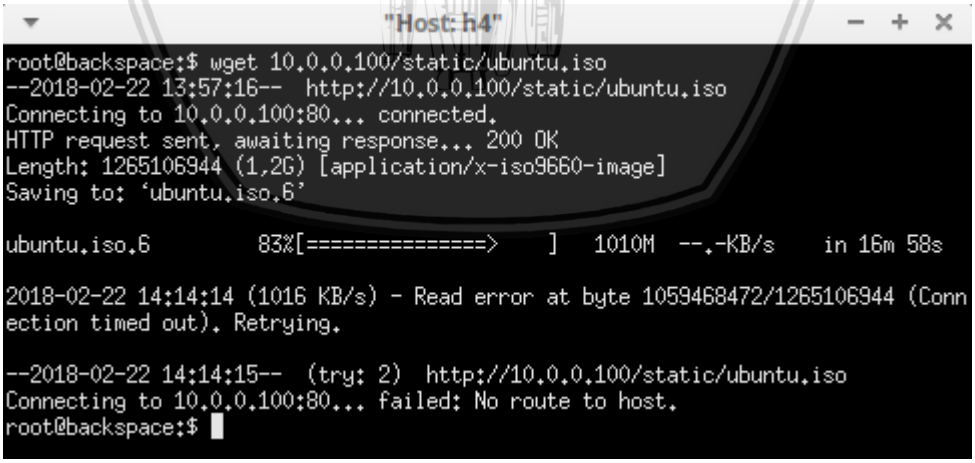


Gambar 6.6 *Controller* Mengarahkan Trafik ke *Server h1*

Pada sisi *Controller* juga berhasil mengarahkan trafik ke *server* dengan jumlah koneksi paling sedikit. Selain itu juga terlihat bahwa informasi jumlah koneksi dari setiap *server* dapat diketahui oleh *Controller*.

Gambar 6.7 *Server h1* Melayani *Request* dari *Client*

Kemudian pada sisi *server* terdapat informasi mengenai *request* dari *Client* dan *server* dapat menanggapinya dengan baik.



Gambar 6.8 *Request Data* dari sisi *Client* Terhenti

Namun, proses unduhan terhenti ketika mencapai 83% dan terdapat read error pada byte ke 1059468472/1265106944. Kemudian *Client* mencoba mengirimkan ulang *request* ke *server* namun *request* tersebut tidak dapat ditanggapi oleh *server* dikarenakan tidak terdapat rute ke *server*.

6.1.3 Pembahasan

Pada pengujian download, didapat bahwa algorima *Least Connection* untuk *load balancing* pada *Software Defined Network* dengan menggunakan metode *Agen Psutils* dapat melakukan proses *download* data berukuran 1,2 GigaBytes tanpa adanya kendala pada proses pengirimannya. Berbeda halnya dengan algoritme *Least Connection* untuk *load balancing* pada *Software Defined Network* dengan menggunakan metode *Flow*. Metode tersebut masih belum efektif ketika melakukan *download* data berukuran 1,2 GigaBytes. Ketika proses pengiriman data berlangsung, terjadi kesalahan dalam proses pengirimannya. Hal ini dikarenakan oleh perpindahan jalur trafik yang disebabkan oleh algoritme *load balancing* tersebut.

Pada metode berbasis *Flow*, lama waktu koneksi terhubung ke *server* ditentukan oleh durasi dari *flow*. Jika waktu dari koneksi terhubung ke *server* melebihi durasi dari *flow*, maka jalur koneksi tersebut akan dialihkan atau diperbarui oleh *load balancer* ke server yang baru. Sehingga dengan mekanisme tersebut mengakibatkan terjadinya kesalahan pada proses pengiriman data. Paket TCP yang seharusnya diarahkan kepada *server* A dialihkan menuju *server* B yang ditentukan oleh load balancer. Kemudian urutan paket TCP tidak sesuai dan diabaikan oleh *server* yang ditunjuk oleh *load balancer* tersebut. Sehingga pengiriman data menjadi terputus pada *sequence* tertentu.

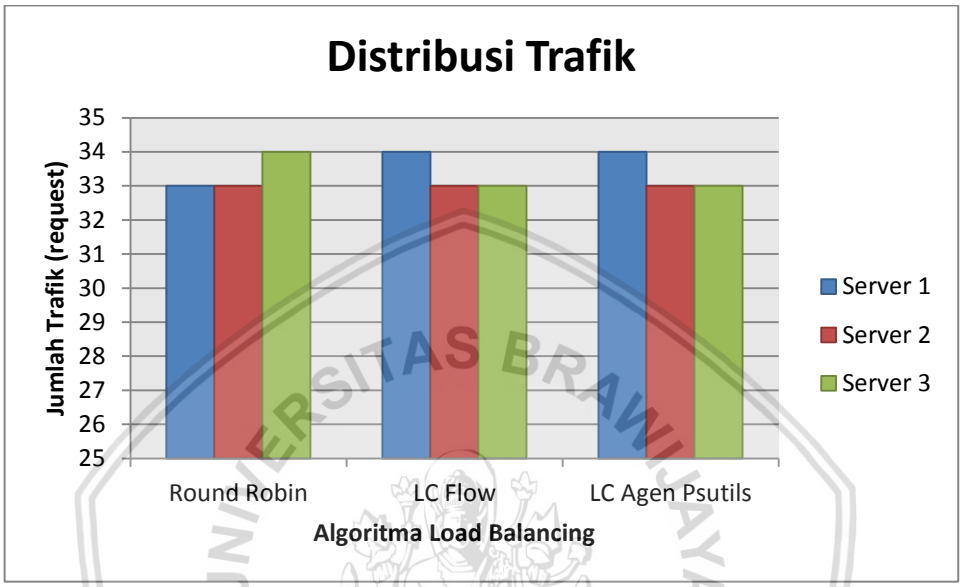
6.2 Pengujian Kinerja

Pengujian pada tahap ini dilakukan bertujuan untuk mengetahui kinerja dari algoritme *least connection* dengan metode *Agen Psutils* yang disertai perbandingan dengan algoritme *round robin* dan *least connection* berbasis *Expired Flow*. Parameter pengujian yang digunakan adalah distribusi trafik, *CPU Utilization*, *Memory Utilization* dan *Response Time*. Pengujian dilakukan menggunakan aplikasi Jmeter, dimana terdapat 3 skenario pengiriman *request* ke *web server* yaitu 100, 200 dan 400 *request*. Pada prosesnya, pengguna akan mengakses 2 *Uniform Resource Locator* (URL) yaitu URL HTML (*http://10.0.0.100/noimage*) dan URL Image (*http://10.0.0.100/static_image*). Pada URL HTML merupakan halaman yang berisi text dengan format HTML, sedangkan URL Image merupakan gambar berukuran 2 MB. *Request* yang dikirimkan oleh pengguna ke 2 halaman yang berbeda ini bertujuan untuk memberikan variasi beban koneksi yang berbeda. Dimana pada URL HTML merupakan beban *request* yang ringan dengan ukuran relatif kecil, sedangkan pada URL Image merupakan beban *request* yang besar.

6.2.1 Skenario Pengiriman 100 *Request*

Pada pengujian ini, dilakukan pengiriman oleh 50 pengguna yang mengakses 2 halaman html dan gambar. Sehingga terdapat 100 *request* yang akan diterima oleh *web server.*

6.2.1.1 *Distribusi Trafik*

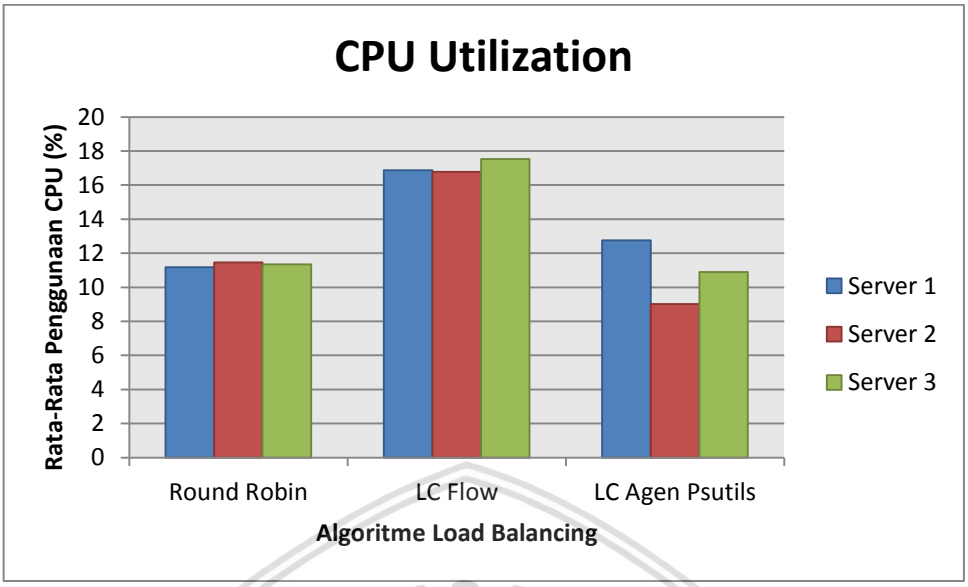


Gambar 6.9 Distribusi Trafik Pengiriman 100 *Request*

Hasil yang didapat dari pengujian 100 *request* terhadap distribusi trafik pada masing-masing algoritme tidak memiliki perbedaan yang signifikan. Masing-masing algoritme mendistribusikan trafik secara merata ke setiap server dialam cluster. Hanya saja pada *round robin*, server ke 3 lebih banyak mendapatkan trafik. Kemudian pada *least connection* berbasis *flow* (*LC Flow*) dan *least connection* berbasis *Agen Psutils* (*LC Agen Psutils*) banyak mendistribusikan trafik ke server 1.

6.2.1.2 *CPU Utilization*

Pada Gambar 6.10 menunjukkan hasil perbandingan rata-rata penggunaan CPU dari setiap *server* pada masing-masing algoritme ketika pengujian dengan pengiriman 10 *request* per detik dilakukan.

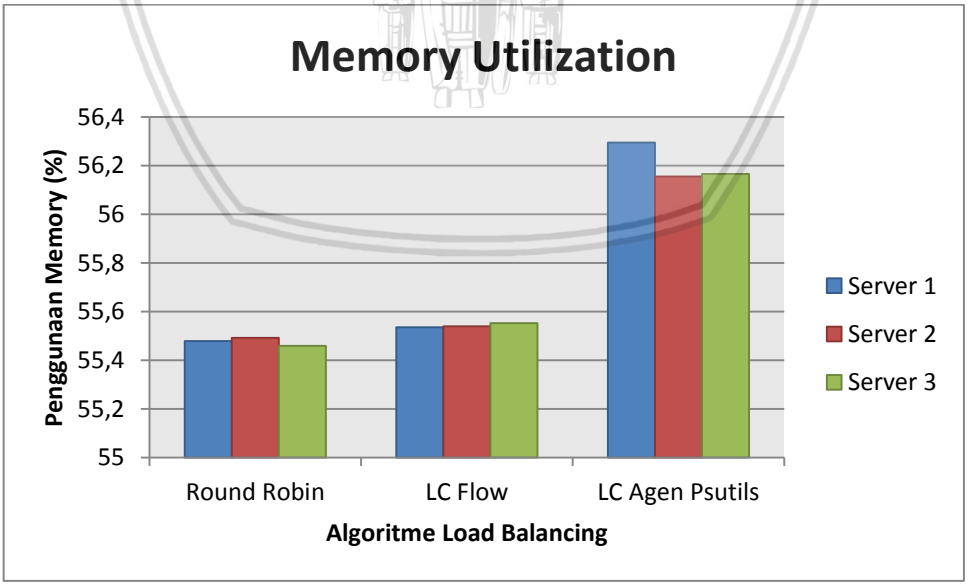


Gambar 6.10 CPU Utilization Pengiriman 100 *Request*

Dari gambar tersebut menunjukkan bahwa algoritme *LC Flow* merupakan algoritme yang penggunaan CPU-nya paling tinggi dibandingkan *round robin* dan *LC Agen Psutils*. Pada *LC Agen Psutils* penggunaan CPU pada masing-masing server sesuai dengan distribusi trafik yang dimiliki.

6.2.1.3 *Memory Utilization*

Gambar 6.11 merupakan rata-rata penggunaan memory untuk setiap *server* pada algoritme *Round Robin, LC Flow* dan *LC Agen Psutils.*



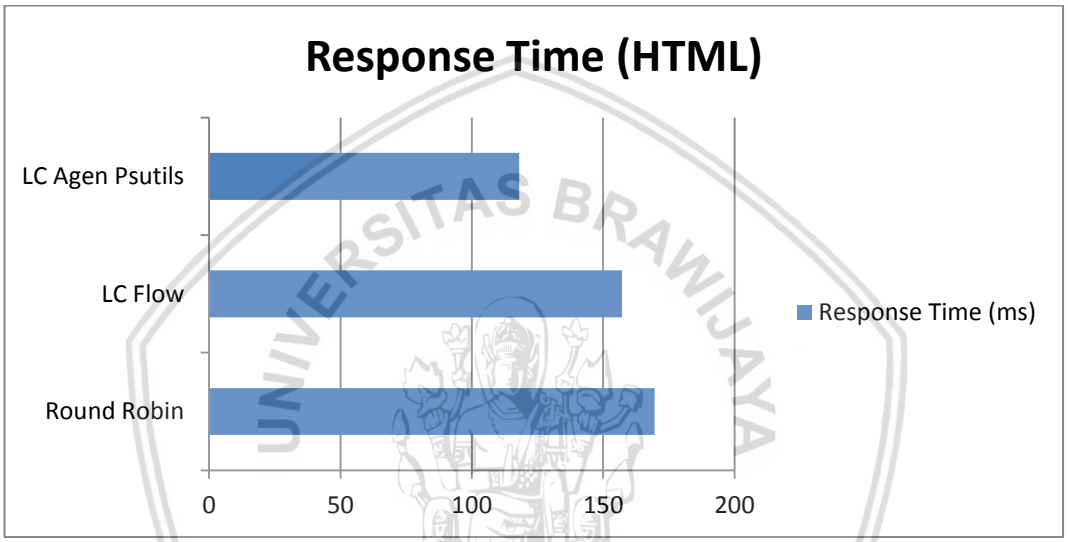
Gambar 6.11 Memory Utilization Pengiriman 100 *Request*

Rata-rata penggunaan memory paling kecil terdapat pada algoritme *Round Robin*. Dimana pada setiap *server* memiliki nilai persentase yang sama yaitu 55,4%. Sedangkan rata-rata penggunaan memory paling besar terdapat pada *LC*

Agen Psutils untuk setiap *server* yaitu 56.29% pada *server* 1, 56.15% pada *server* 2 dan 56.16% pada *server* 3.

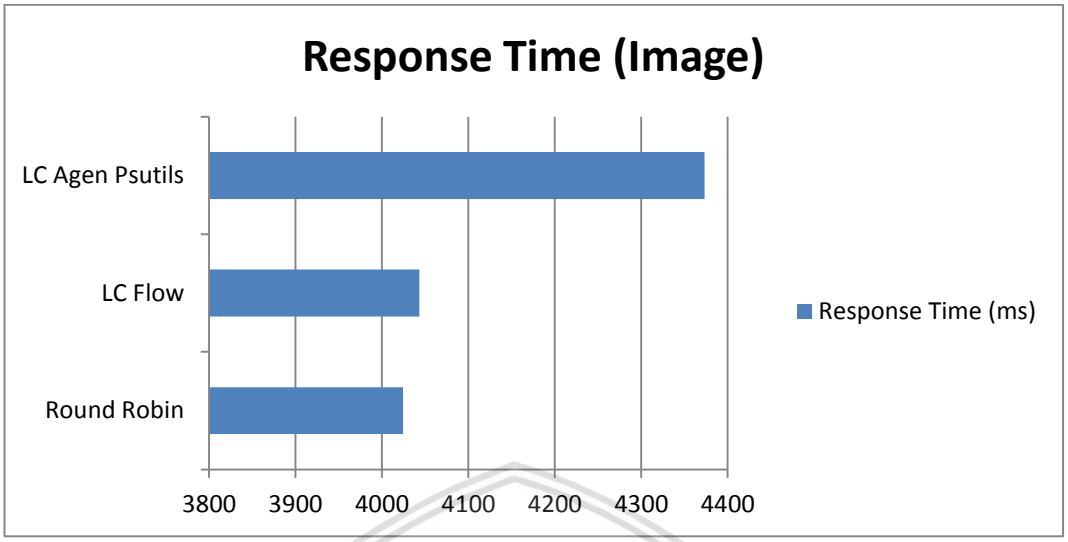
6.2.1.4 *Response Time*

Pada pengujian untuk mendapatkan data *response time*, penelitian ini membagi data response time menjadi 2 bagian yaitu data response time untuk request pada halaman HTML dan Image. Kemudian dari masing-masing bagian tersebut ditentukan rata-rata response time dari keseluruhan request yang diterima oleh server. Sehingga dari skenario tersebut didapatlah hasil seperti pada dan 6.12 dan 6.13.



Gambar 6.12 Response Time HTML Pengiriman 100 *Request*

Dari pengujian yang telah dilakukan, didapat data response time dari request terhadap halaman HTML untuk algoritme *Round Robin, LC Flow* dan *LC Agen Psutils.* Dari grafik pada Gambar 6.12, algoritme *Round Robin* memiliki nilai response time tertinggi yaitu 169.9 ms. Sedangkan yang paling rendah adalah algoritme *LC Agen Psutils* yaitu 118.09 ms.



Gambar 6.13 Response Time Image Pengiriman 100 Request

Gambar 6.13 merupakan grafik response time untuk request pada halaman Image. Dari grafik menunjukkan bahwa algoritme LC Agen Psutils memiliki waktu tanggap yang lebih tinggi terhadap request pada halaman Image yaitu 8746.8 ms.

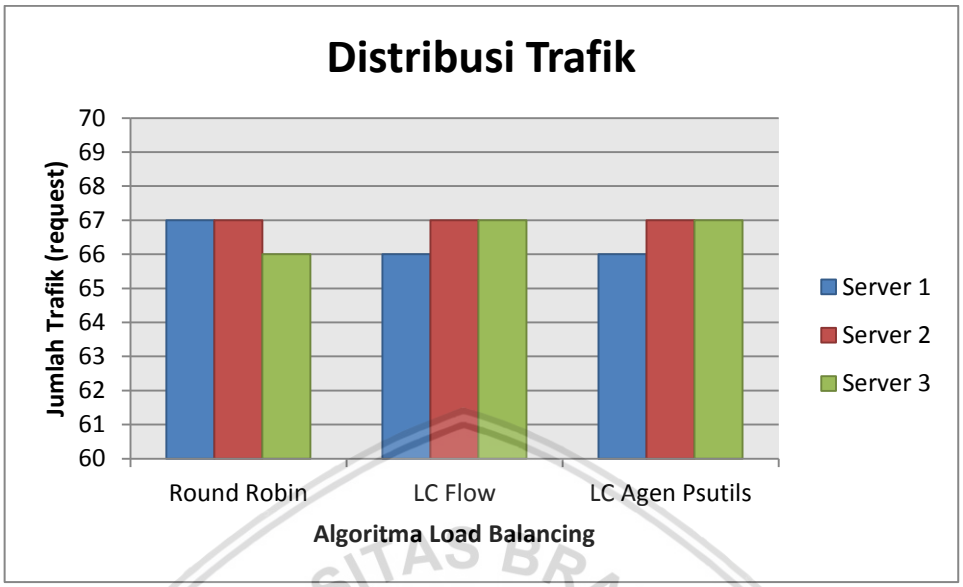
Dari grafik pada 6.12 dan 6.13 dapat diketahui bahwa algoritme LC Agen Psutils memiliki waktu tanggap yang lebih cepat terhadap request pada halaman HTML dan memiliki waktu tanggap yang lebih lambat terhadap request pada halaman Image.

6.2.2 Skenario Pengiriman 200 *Request*

Pada pengujian ini, dilakukan pengiriman oleh 100 pengguna yang mengakses 2 halaman html dan gambar. Sehingga terdapat 200 *request* yang akan diterima oleh *web server.*

6.2.2.1 *Distribusi Trafik*

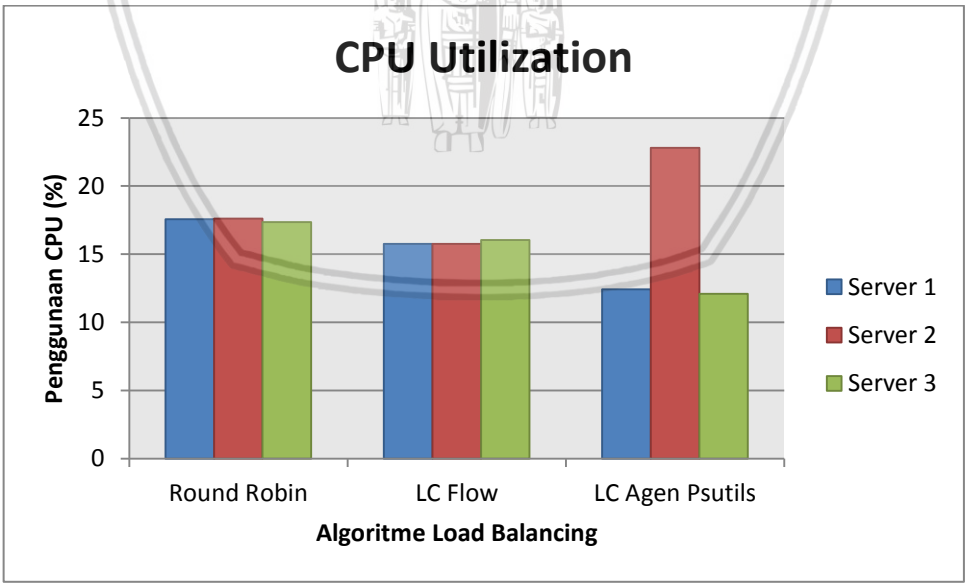
Distribusi trafik untuk algoritme *round robin*, *LC Flow* dan *LC Agen Psutils* pada pengiriman 200 *request* dapat ditunjukkan pada Gambar 6.14.



Gambar 6.14 Distribusi Trafik Pengiriman 200 *Request*

Pendistribusian trafik pada *LC Agen Psutils* pada pengiriman 200 *request* ke server mengalami perubahan dengan pengujian pengiriman 100. Dimana pada algoritme *LC Agen Psutils,* trafik terbanyak dialokasikan ke *server* 2 dengan jumlah sebanyak 68 dari 200 trafik.

6.2.2.2 *CPU Utilization*

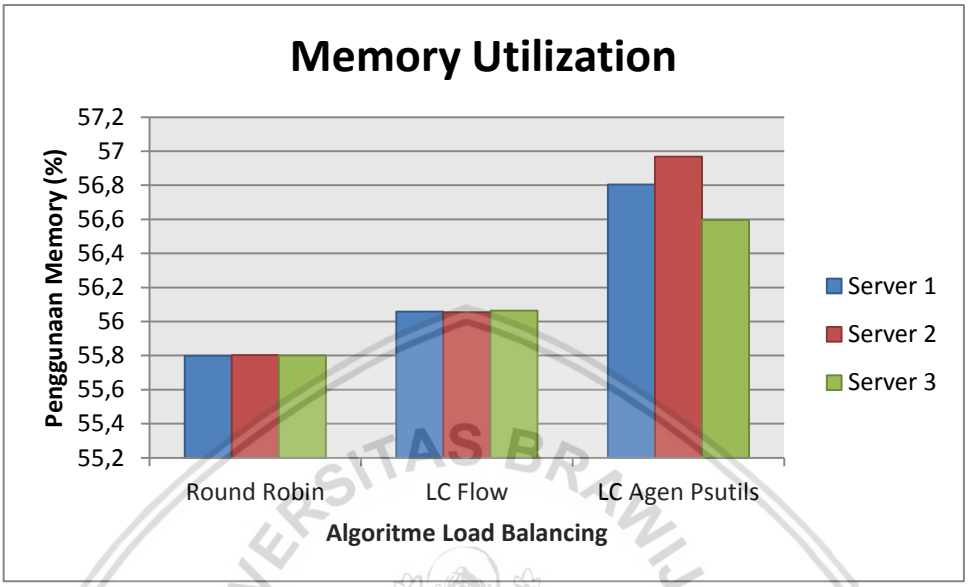


Gambar 6.15 CPU Utilization Pengiriman 200 *Request*

Pada Gambar 6.14, *server* dengan penggunaan CPU tertinggi pada *LC Agen Psutils* masih dimiliki oleh *server* 2. Hal ini berbeda dari skenario sebelumnya (skenario pengiriman 100 *request*). Hal itu dikarenakan distribusi trafik banyak

dialokasikan ke server 2, sehingga penggunaan CPU pada server menjadi meningkat.

6.2.2.3 *Memory Utilization*

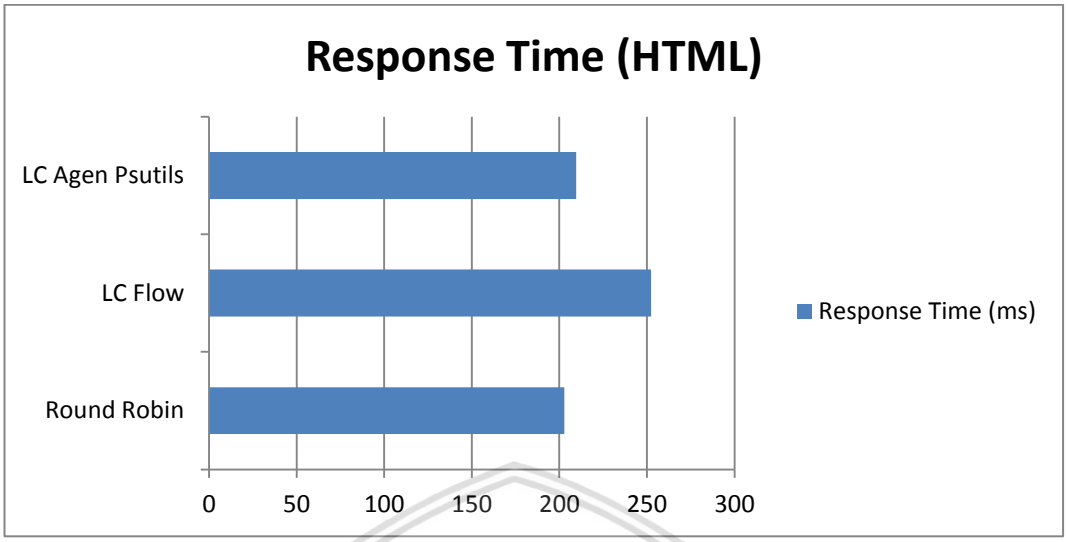


Gambar 6.16 Memory Utilization Pengiriman 200 *Request*

Pada Gambar 6.16 menunjukkan data rata-rata penggunaan Memory pada skenario pengiriman 20 *request* per detik. Perubahan yang signifikan dialami oleh setiap algoritme. Dimana pada pengujian sebelumnya (skenario pengiriman 10 *request* per detik) *Round Robin* mendapatkan nilai terkecil untuk rata-rata penggunaan memory. Sedangkan pada skenario ini nilai yang didapat mengalami peningkatan. Pada *LC Flow*, rata-rata penggunaan memory pada *server* 1 mengalami peningkatan yang drastis. Sedangkan LC *Agen Psutils* mengalami penurunan pada skenario pengiriman 20 *request* per detik.

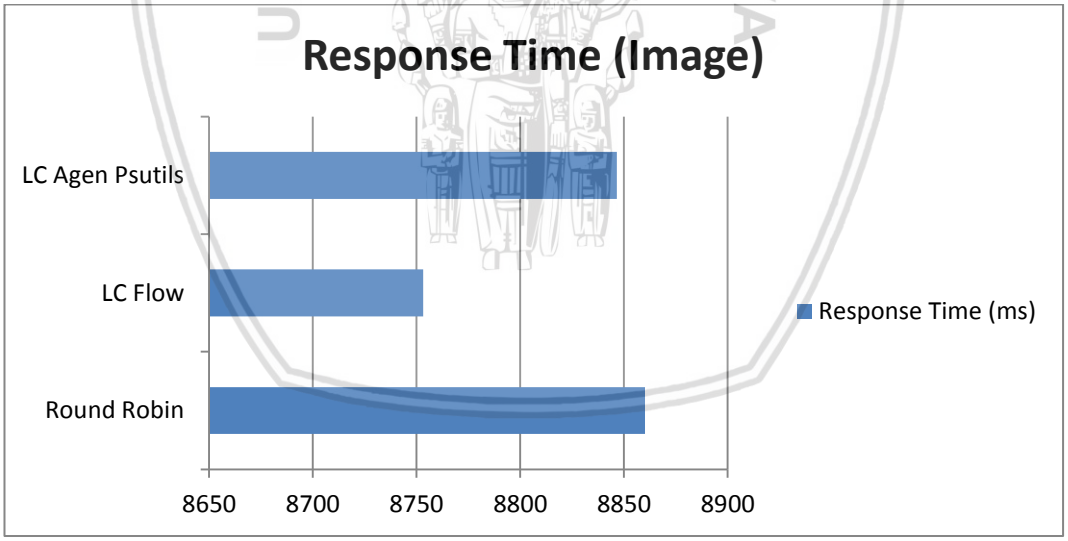
6.2.2.4 *Response Time*

Pada pengujian untuk mendapatkan data *response time*, penelitian ini membagi data response time menjadi 2 bagian yaitu data response time untuk request pada halaman HTML dan Image. Kemudian dari masing-masing bagian tersebut ditentukan rata-rata response time dari keseluruhan request yang diterima oleh server. Sehingga dari skenario tersebut didapatlah hasil seperti pada dan 6.18 dan 6.19.



Gambar 6.17 Response Time HTML Pengiriman 200 *Request*

Pada skenario pengiriman 200 request ini, didapat data response time dari request terhadap halaman HTML untuk algoritme *Round Robin, LC Flow* dan *LC Agen Psutils.* Dari grafik pada Gambar 6.18, algoritme *LC Flow* memiliki nilai response time tertinggi yaitu 252.355 ms. Sedangkan yang paling rendah adalah algoritme *Round Robin* yaitu 202.87 ms.



Gambar 6.18 Response Time Image Pengiriman 200 *Request*

Pada Gambar 6.19 adalah grafik waktu tanggap untuk request oleh client pada halaman Image. Dari grafik memperlihatkan bahwa algoritme *Round Robin* memiliki waktu tanggap yang lebih tinggi terhadap request pada halaman Image yaitu 8860.095 ms.

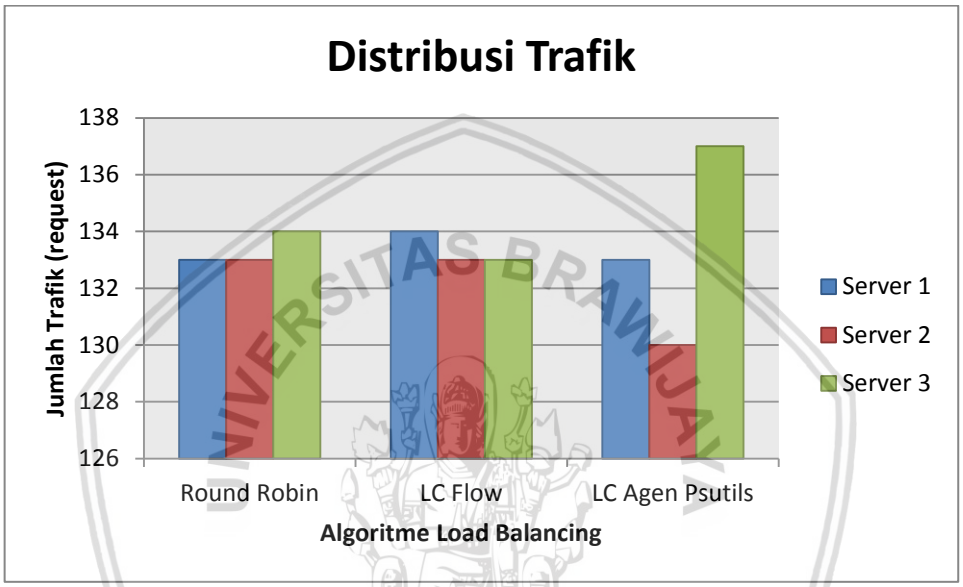
Dari grafik pada Gambar 6.18 dan 6.19 dapat diketahui bahwa waktu tanggap oleh web server untuk algoritme *LC Agen Psutils*, *LC Flow* dan *Round Robin* meningkat pada request dari client sebanyak 200.

6.2.3 Skenario Pengiriman 400 *Request*

Pada pengujian ini, dilakukan pengiriman oleh 200 pengguna yang mengakses 2 halaman html dan gambar. Sehingga terdapat sejumlah 400 *request* yang akan diterima oleh *web server.*

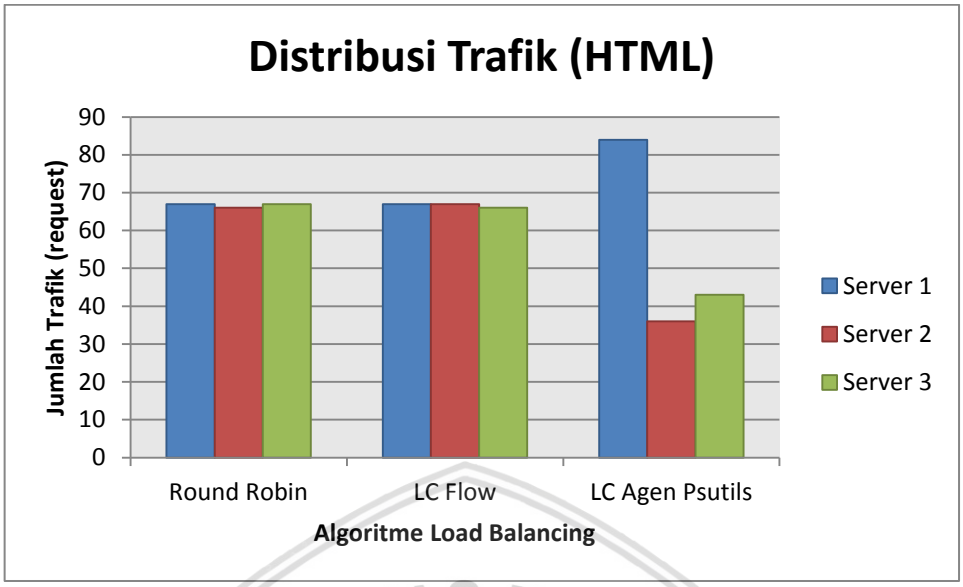
6.2.3.1 *Distribusi Trafik*

Distribusi trafik untuk algoritme *round robin*, *LC Flow* dan *LC Agen Psutils* pada pengiriman 400 *request* dapat ditunjukkan pada Gambar 6.19.



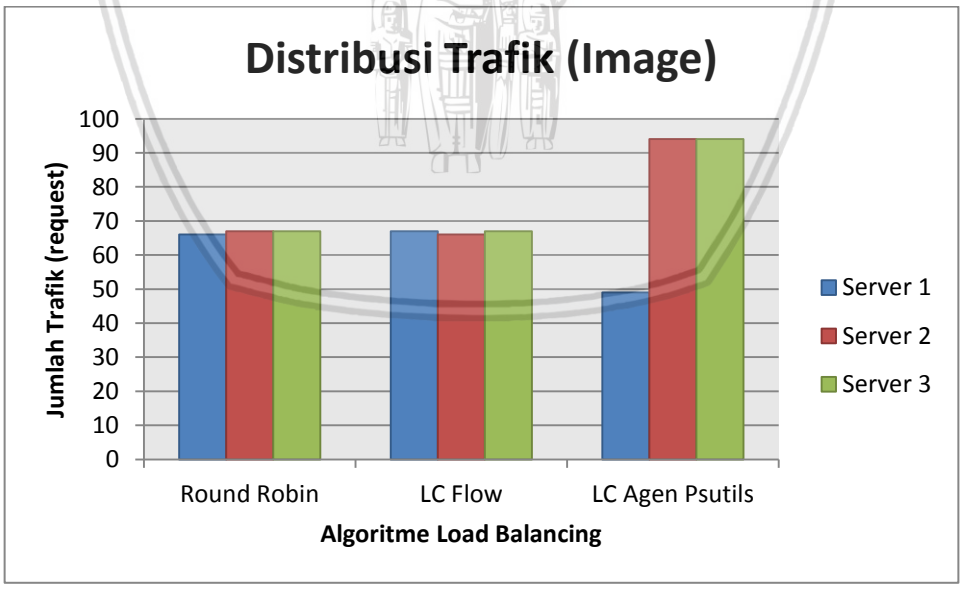
Gambar 6.19 Distribusi Trafik Pengiriman 400 *Request*

Distribusi trafik oleh *LC Agen Psutils* pada skenario pengiriman 400 *request* ini memiliki perbedaan dari skenario pengiriman 100 dan 200 *request*. Dimana pada skenario tersebut pendistribusian trafik terlihat lebih stabil, sedangkan pada skenario ini *server* 3 mendapatkan trafik yang lebih banyak dibandingkan *server* 1 dan 2. Dari 400 trafik yang terkirim, *server* 1 mendapatkan 133 *request*, server 2 mendapatkan 130 *request* dan server 3 mendapatkan 137 *request*. Grafik distribusi trafik pada skenario ini terlihat tidak stabil disebabkan banyaknya *request* yang ditangani oleh server dan variasi *request* yang dilakukan oleh client terhadap halaman HTML dan Image. Dari 500 jumlah trafik, client mengakses halaman HTML sebanyak 250 dan halaman Image sebanyak 250. Pada *request* terhadap halaman Image, proses *request* berlangsung lama. Karena halaman tersebut memiliki ukuran data yang besar yaitu 2 MB.



Gambar 6.20 Distribusi Trafik HTML 400 *Request*

Gambar 6.20 merupakan grafik jumlah trafik untuk halaman HTML yang ditangani oleh setiap server. Pada algoritme *LC Agen Psutils*, server 1 banyak menangani trafik terhadap halaman HTML. Sedangkan pada algoritme round robin dan LC Flow, trafik untuk halaman HTML dibagikan secara merata ke setiap server.



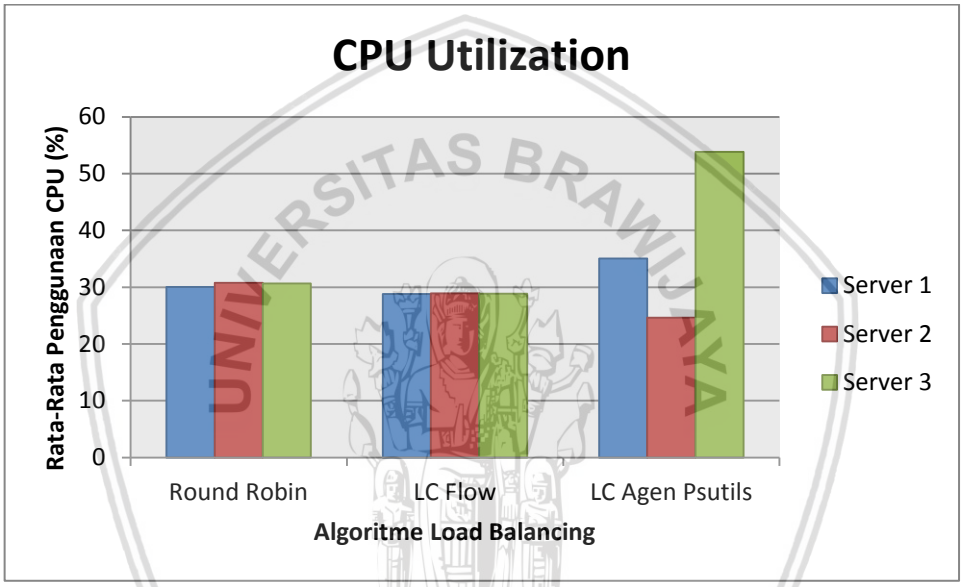
Gambar 6.21 Distribusi Trafik Image 400 *Request*

Gambar 6.21 merupakan distribusi trafik pada server terhadap halaman *Image*. Pada algoritme *LC Agen Psutils*, server 2 dan 3 mendapatkan trafik untuk halaman *Image* lebih banyak dibandingkan server 1. Sedangkan pada algoritme

round robin dan LC Flow, trafik untuk halaman Image dibagi merata ke setiap server.

Dari kedua data pada Gambar 6.20 dan 6.21, dapat ketahui bahwa algoritme *LC Agen Psutils* mampu mendistribusikan trafik berdasarkan beban koneksi yang dimiliki oleh setiap server. *Request* yang dilakukan client terhadap halaman *Image* memiliki waktu koneksi yang lebih lama dibandingkan halaman HTML karena besarnya beban dari halaman tersebut. Sehingga server yang melayani *request* terhadap halaman Image, akan mendapatkan *request* yang sedikit untuk halaman HTML.

6.2.3.2 *CPU Utilization*

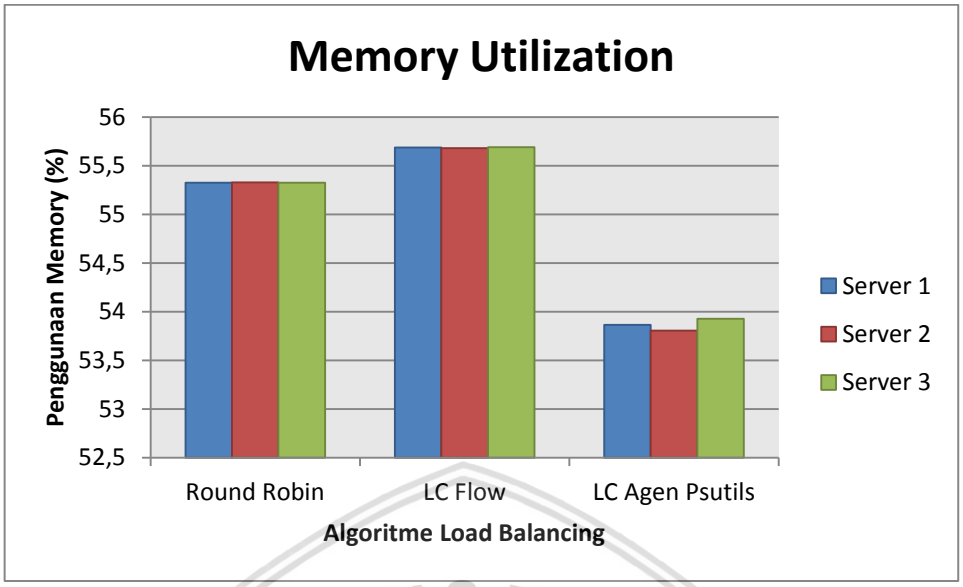


Gambar 6.22 CPU Utilization Pengiriman 400 *Request*

Pada skenario pengujian dengan pengiriman 400 *request* untuk mendapatkan data rata-rata penggunaan CPU ini algoritme LC Flow memiliki rata-rata penggunaan CPU yang paling rendah. Sedangkan algoritme *LC Agen Psutils* memiliki rata-rata penggunaan CPU yang paling tinggi. Khususnya pada server ke 3. Hal ini dikarenakan pada server tersebut banyak menerima *request* terhadap URL Image. Beban *request* yang besar menyebabkan penggunaan CPU pada server tersebut meningkat.

6.2.3.3 *Memory Utilization*

Gambar 6.23 merupakan rata-rata penggunaan memory untuk setiap *server* pada algoritme *Round Robin, LC Flow* dan *LC Agen Psutils.*

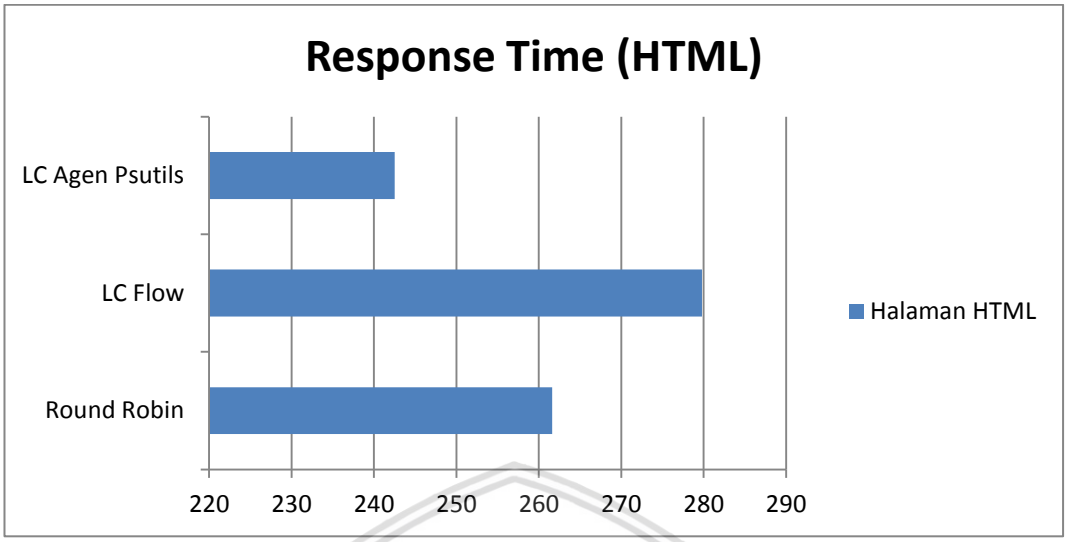


Gambar 6.23 Memory Utilization Pengiriman 400 *Request*

Rata-rata penggunaan memory paling kecil terdapat pada algoritme *LC Agen Psutils*. Dimana pada setiap *server* memiliki nilai persentase yang sama yaitu 53%. Sedangkan rata-rata penggunaan memory paling besar terdapat pada *LC Flow* untuk setiap *server* yaitu 55.68% untuk setiap server. Server 3 pada algoritme *LC Agen Psutils* meningkat dikarenakan pada server tersebut banyak menerima *request* terhadap URL Image seperti yang telah dibahas di skenario pengiriman 200 *request* pada CPU Utilization.

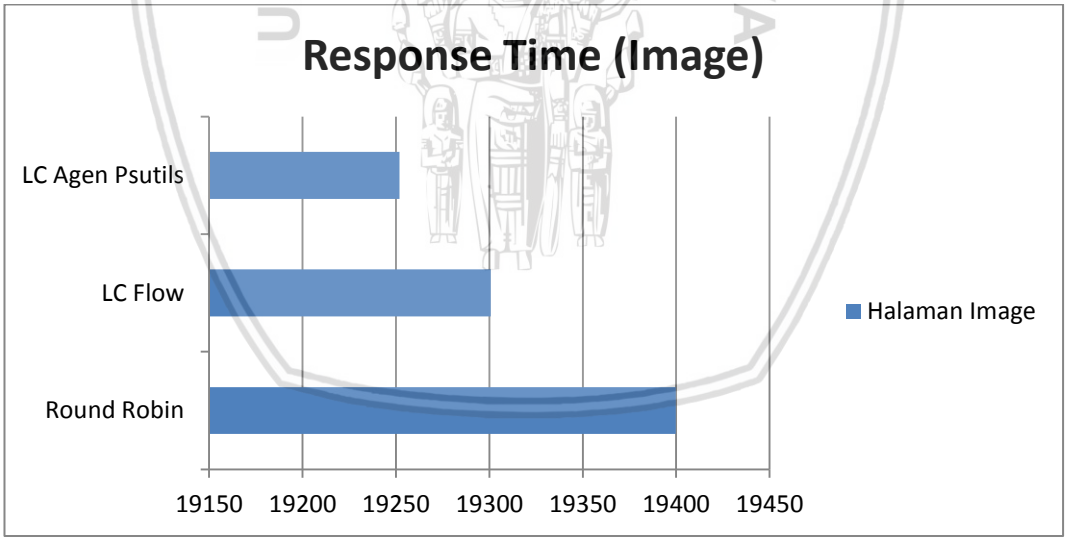
6.2.3.4 *Response Time*

Pada pengujian untuk skenario pengiriman 400 request ini data response time menjadi 2 bagian yaitu data response time untuk request pada halaman HTML dan Image. Kemudian dari masing-masing bagian tersebut ditentukan rata-rata response time dari keseluruhan request yang diterima oleh server. Sehingga dari skenario tersebut didapatlah hasil seperti pada dan 6.25 dan 6.26.



Gambar 6.24 Response Time HTML Pengiriman 400 *Request*

Pada skenario pengiriman 500 request ini, didapat nilai response time dari request terhadap halaman HTML untuk algoritme *Round Robin, LC Flow* dan *LC Agen Psutils*. Algoritme LC Agen Psutils mendapatkan nilai response time paling rendah yaitu 242.515 ms. Sedangkan algoritme LC Flow memiliki waktu tanggap yang lebih lama dibandingkan kedua algoritme yaitu 279.815 ms.



Gambar 6.25 Response Time Image Pengiriman 400 *Request*

Gambar 6.26 merupakan grafik response time untuk request pada halaman Image. Dari grafik menunjukkan bahwa algoritme LC Agen Psutils memiliki waktu tanggap yang lebih tinggi terhadap request pada halaman Image yaitu 8746.8 ms.

Dari grafik pada 6.25 dan 6.26 dapat diketahui bahwa algoritme LC Agen Psutils memiliki waktu tanggap yang lebih cepat terhadap request pada halaman HTML maupun terhadap request pada halaman Image.

6.2.4 Pembahasan

Dari pengujian kinerja dengan skenario pengiriman 100, 200 dan 400 *request* dapat ditemukan bahwa pendistribusian trafik oleh algoritme *least connection* berbasis *Agen Psutils* (*LC Agen Psutils*) terlihat tidak stabil seiring bertambahnya jumlah *request.* Hal ini dikarenakan pendistribusian trafik berdasarkan jumlah koneksi pada server. Sehingga perbedaan beban *request* yang diterima oleh server akan mempengaruhi jumlah trafik yang diterima oleh setiap server. Server yang banyak menerima *request* ke URL Image memiliki waktu koneksi yang lebih lama. Sedangkan server yang banyak menerima *request* ke URL HTML memiliki waktu koneksi yang lebih cepat.

Selanjutnya, algoritme *LC Agen Psutils* memiliki penggunaan CPU yang berbanding lurus terhadap jumlah trafik yang diterima oleh masing-masing *server*. *Server* yang melayani trafik terbanyak akan menggunakan sumber daya yang banyak pula. Selain itu, algoritme ini juga memiliki penggunaan CPU yang tertinggi dibandingkan algoritme *round robin* dan *LC Flow*. Hal ini disebabkan oleh program *Agen Psutils* yang terpasang pada setiap server. Program tersebut mengirimkan jumlah koneksi ke *SDN Controller* secara terus menerus sehingga meningkatkan penggunaan CPU pada server.

Algoritme *LC Agen Psutils* memiliki nilai rata-rata *response time* yang lebih kecil dibandingkan algoritme *round robin* dan *LC Flow.* Pada skenario 100 *request*, nilai response time dari LC Agen Psutils adalah 118,0 ms sedangkan *LC Flow* dan *Round Robin* adalah 157,2 ms dan 169,7 ms. Kemudian pada skenario 200 *request*, nilai response time dari LC Agen Psutils adalah 202,8 ms sedangkan LC Flow dan Round Robin adalah 252,3 ms dan 209,5 ms. Selanjutnya pada skenario 400 *request*, nilai *response time* dari *LC Agen Psutils* adalah 261,61 ms sedangkan *LC Flow* dan *Round Robin* adalah 279,8 ms dan 242,5 ms. Kecilnya response time yang dimiliki oleh algoritme *LC Agen Psutils* ini disebabkan oleh skema pendistribusian trafik yang dinamis. *Request* yang masuk akan diarahkan ke server dengan beban koneksi yang sedikit. Sehingga server tersebut akan dapat melayani *request* lebih cepat. Sedangkan algoritme *round robin,* akan mengirimkan trafik ke server tanpa melihat beban yang dimiliki oleh server tersebut. Kemudian pada algoritme *LC Flow*, trafik akan dikirimkan ke server berdasarkan jumlah flow yang mengarah ke server tersebut. Server dengan jumlah flow sedikit yang akan menerima trafik baru. Namun setiap flow memiliki waktu timeout yang sama. Baik itu untuk URL HTML maupun URL Image. Oleh karena itu, perlakuan algoritme *LC Flow* terhadap trafik hampir mirip dengan algoritme *round robin* yang bersifat statis.

BAB 7 PENUTUP

Dari beberapa tahap yang telah dilakukan pada penelitian ini, sampailah pada tahap akhir dari penelitian. Berikut adalah kesimpulan yang dapat diambil dan saran yang dapat digunakan untuk penelitian-penelitian selanjutnya.

7.1 Kesimpulan

Kesimpulan yang dapat diambil dari penelitian dengan judul "Implementasi Load Balancing Menggunakan Algoritme *Least Connection* dengan *Agen Psutils* pada *Web Server*" ini mencakup beberapa hal yaitu hasil implementasi algoritme *load balancing* pada *Software Defined Network* (SDN), pengujian *download* data, pengujian kinerja algoritme dengan parameter distribusi trafik, *CPU utilization, memory utilization* dan *response time.*

Pada implementasi algoritme *load balancing*, *SDN Controller* dijadikan sebagai *load balancer* yang menerapkan algoritme *Least Connection*. Controller yang digunakan adalah POX Controller. Kemudian terdapat sebuah agen yang disebut *Agen Psutils* yang berjalan pada setiap server. Agen tersebut mengirimkan informasi jumlah koneksi yang terjalin antara *client* dan server, sehingga *SDN Controller* dapat mengetahui jumlah koneksi yang dimiliki oleh setiap server. Dari implementasi tersebut, teknologi *load balancing* menggunakan algoritme *Least Connection* dengan *Agen Psutils* pada *Software Defined Network* (SDN) dapat mendistribusikan trafik ke beberapa *server* berdasarkan jumlah koneksi aktif yang dimiliki oleh setiap *server*.

Setelah implementasi algoritme *load balancing* pada SDN berjalan dengan baik, dilakukan pengujian *download* data. Dari pengujian tersebut didapat hasil bahwa algoritme *Least Connection* dengan *Agen Psutils* (*LC Agen Psutils*) mampu mengatasi kekurangan dari algoritme *Least Connection* berbasis *Expired Flow* (*LC Flow*). Algoritme *LC Agen Psutils* dapat melakukan proses unduhan data berukuran 1,2 GigaBytes tanpa ada kendala dalam proses pengirimannya. Sedangkan pada algoritme *LC Flow,* proses unduhan data terhenti pada proses ke 83% ketika data dikirimkan.

Dari keseluruhan skenario pengujian kinerja terhadap algoritme *Round Robin, Least Connection* berbasis Expired F*low (LC Flow)* dan *Least Connection* berbasis *Agen Psutils (LC Agen Psutils).* Berikut ini beberapa hal yang dapat disimpulkan adalah sebagai berikut:

1. Algoritme *Least Connection* dengan *Agen Psutils* memiliki nilai response time yang lebih kecil jika dibandingkan dengan algoritme *Round Robin* dan *Least Connection* dengan *Expired Flow*. Pada skenario 400 *request*, nilai *response time* dari *LC Agen Psutils* adalah 261,61 ms sedangkan *LC Flow* dan *Round Robin* adalah 279,8 ms dan 242,5 ms. Hal ini dikarenakan trafik yang datang ke jaringan akan diarahkan ke server dengan beban koneksi

yang sedikit. Sehingga akan mempercepat proses pengiriman data ke client.

2. Ukuran pengiriman data mempengaruhi penggunaan *CPU*, *Memory* dan *response time* pada server. Hal ini dapat dibuktikan dengan hasil pengujian terhadap *request* URL HTML dengan ukuran 153 Bytes dan *request* URL Image dengan ukuran 1,93 MegaBytes.
3. Algoritme *Least Connection* dengan *Agen Psutils* dapat mendistribusikan trafik berdasarkan jumlah koneksi yang dimiliki oleh server tersebut.
4. Semakin banyak *request* yang diterima oleh server, maka semakin tinggi penggunaan *CPU* dan *Memory* pada *server*.
5. Semakin banyak *request* yang diterima oleh server, maka *response time* dari client ke server semakin meningkat.
6. Algoritme *Round Robin* dan *LC Flow* memiliki grafik distribusi trafik yang lebih stabil jika dibandingkan dengan *LC Agen Psutils.*
7. Tidak terjadi error pada pengujian pengiriman data HTML dan pengiriman data *Image* dengan skenario pengiriman 100, 200 dan 400 *request* per detik.

7.2 Saran

Dari implementasi dan pengujian yang telah dilakukan pada penelitian ini, adapun saran yang dapat digunakan untuk pengembangan pada penelitian selanjutnya, antara lain:

1. Dapat menerapkan sistem *load balancing* pada perangkat keras *(dedicated switch)* yang mendukung protokol *OpenFlow*.
2. Dapat melakukan pengujian dengan jumlah *request* yang lebih tinggi.
3. Dapat melakukan optimasi pada program *Agen Psutils* sehingga dapat mengurangi penggunaan *CPU* dan *Memory* pada server.
4. Dapat menerapkan metode untuk menentukan jumlah koneksi dari server hanya dari SDN Controller tanpa menggunakan *Agen*.

DAFTAR PUSTAKA

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

Apache JMeter. (2018). *Apache JMeter*. Retrieved July 10, 2018, from Apache JMeter - User's Manual: Elements of a Test Plan: https://jmeter.apache.org/usermanual/test_plan.html

Dooley, K. (2002). Designing Large-scale LANs. In K. Dooley, *Designing Large-scale LANs.* O'Reilly.

Erine, K. (2016). Analisis Openflow Load Balancing Webserver dengan Algoritma Least Connection Pada Software Defined Network.

Julianto, R. (2016). Implementasi Load Balancing Menggunakan Metode Berbasis Sumber Daya CPU pada Software Defined Networking.

Kaur & Singh. (2015). Round-Robin Based Load Balancing in Software Defined Networking. SBS State Technical Campus.

Lantz, B. (2015). A Mininet-based Virtual Testbed for Distributed SDN Development.

Lukitasari, D. (2010). Analisis Perbandingan Load Balancing Web Server Tunggal dengan Web server Cluster Menggunakan Linux Virtual Server.

Mahmood, A., & Rashid, I. (2011). Comparison of Load Balancing Algorithms for Clustered Web Servers.

McCauley, M. (2015, Maret 5). *POX Wiki*. Retrieved Desember 18, 2017, from https://openflow.stanford.edu/display/ONL/POX+Wiki

Mosberger. (1998). httperf - A Tool for Measuring Web Server Performance.

Mustafa, E. (2015). Load Balancing Algorithms Round-Robin, Least-Connection And Least Loaded Efficiency. Shaqra University.

Nadeau & Gray. (2013). *Software Defined Networks.* O'Reilly.

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

Pressman. (2002). Rekayasa Perangkat Lunak: Pendekatan Praktisi (Buku I).

Sirajuddin & Affandi. (2012). Rancang Bangun Server Learning Management System Menggunakan Load Balancer dan Reverse Proxy.

LAMPIRAN DATA HASIL PENGUJIAN

7.1 Skenario Pengiriman 100 *Request*

7.1.1 Distribusi Trafik

Algoritme	Server 1	Server 2	Server 3	Total
Round Robin	33	33	34	100
LC Flow	34	33	33	100
LC Agen Psutils	34	33	33	100

7.1.2 CPU Utilization

Algoritme	Server 1	Server 2	Server 3
Round Robin	11,17576	11,46061	11,34412
LC Flow	16,86765	16,77576	17,53333
LC Agen Psutils	12,74706	9,009091	10,88788

7.1.3 Memory Utilization

Algoritme	Server 1	Server 2	Server 3
Round Robin	55,47879	55,49091	55,45882
LC Flow	55,53529	55,53939	55,55152
LC Agen Psutils	56,29394	56,15455	56,16471

7.1.4 Response Time

Algoritme	Response Time (ms)
Round Robin	169,6
LC Flow	157,2
LC Agen Psutils	118,09

Algoritme	Response Time (ms)
Round Robin	4024,14
LC Flow	4043,06
LC Agen Psutils	4373,29

7.2 Skenario Pengiriman 200 *Request*

7.2.1 Distribusi Trafik

Algoritme	Server 1	Server 2	Server 3	Total
Round Robin	67	67	66	200
LC Flow	66	67	67	200
LC Agen Psutils	66	67	67	200

7.2.2 CPU Utilization

Algoritme	Server 1	Server 2	Server 3
Round Robin	17,55672	17,60597	17,35909
LC Flow	15,75	15,76418	16,02836
LC Agen Psutils	12,42727	22,80896	12,09851

7.2.3 Memory Utilization

Algoritme	Server 1	Server 2	Server 3
Round Robin	55,79851	55,80299	55,8
LC Flow	56,05909	56,05522	56,06418
LC Agen Psutils	56,80455	56,96866	56,59552

7.2.4 Response Time

Algoritme	Response Time (ms)
Round Robin	202,87
LC Flow	252,355
LC Agen Psutils	209,57

Algoritme	Response Time (ms)
Round Robin	8860,095
LC Flow	8753,225
LC Agen Psutils	8846,525

7.3 Skenario Pengiriman 400 *Request*

7.3.1 Distribusi Trafik

Algoritme	Server 1	Server 2	Server 3	Total
Round Robin	133	133	134	400
LC Flow	134	133	133	400
LC Agen Psutils	133	130	137	400

7.3.2 CPU Utilization

Algoritme	Server 1	Server 2	Server 3
Round Robin	30,057143	30,7406015	30,6283582
LC Flow	28,793284	28,8879699	28,8706767
LC Agen Psutils	35,077444	24,5953846	53,8218978

7.3.3 Memory Utilization

Algoritme	Server 1	Server 2	Server 3
Round Robin	55,32406	55,3270677	55,3246269
LC Flow	55,685075	55,6804511	55,6879699
LC Agen Psutils	53,86391	53,8038462	53,9248175

7.3.4 Response Time

Algoritme	Halaman HTML
Round Robin	261,615
LC Flow	279,815
LC Agen Psutils	242,515

Algoritme	Halaman Image
Round Robin	19399,8175
LC Flow	19300,6725
LC Agen Psutils	19251,935