

**IMPLEMENTASI *INFRASTRUCTURE AS A SERVICE* PADA *CLOUD*
COMPUTING MENGGUNAKAN METODE *LOAD BALANCING***

SKRIPSI

Jurusan Informatika
Program Studi Sarjana Informatika

Oleh :

AGUS SAPUTRA

NIM D1041141049



**FAKULTAS TEKNIK
UNIVERSITAS TANJUNGPURA
PONTIANAK
2020**

HALAMAN PERNYATAAN

Yang bertanda tangan di bawah ini:

Nama : Agus Saputra

NIM : D1041140149

menyatakan bahwa dalam skripsi yang berjudul “Implementasi *Infrastructure as a Service* pada *Cloud Computing* menggunakan *Load Balancing*” tidak terdapat karya yang pernah diajukan untuk memperoleh gelar sarjana di suatu perguruan tinggi manapun. Sepanjang pengetahuan saya, tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Demikian pernyataan ini dibuat dengan sebenar-benarnya. Saya sanggup meneriam konsekuensi akademis dan hukum dikemudian hari apabila pernyataan yang dibuat ini tidak benar.

Pontianak, 2 Februari 2020

Agus Saputra

NIM. D1041140149



KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS TANJUNGPURA
FAKULTAS TEKNIK

Jalan Prof. Dr. H. Hadari Nawawi Pontianak 78124

Telepon (0561) 740186 Faximili (0561) 740186

Email : ft@untan.ac.id Website : teknik.untan.ac.id

HALAMAN PENGESAHAN

**IMPLEMENTASI INFRASTRUCTURE AS A SERVICE PADA CLOUD
COMPUTING MENGGUNAKAN METODE LOAD BALANCING**

Jurusan Informatika
Program Studi Sarjana Informatika

Oleh :

Agus Saputra
NIM. D1041141049

Telah dipertahankan di depan Penguji Skripsi pada tanggal 28 Januari 2020
dan diterima sebagai salah satu persyaratan untuk memperoleh gelar sarjana.

Susunan Penguji Skripsi:

Ketua,

Heri Priyanto, S.T., M.T.
NIP. 197504122003121001

Penguji Utama,

Helfi Nasution, S.Kom., M.Cs.
NIP. 197104291998021001

Sekretaris,

Novi Safriadi, S.T., M.T.
NIP. 198604302012122002

Penguji Pendamping,

M. Azhar Irwansyah, S.T., M.Eng.
NIP. 198506062008121002

Pontianak, 28 Januari 2020
Dekan,



Dr. rer.nat. Ir. R. M. Rustamaji, M.T.
NIP. 196801161994031003

HALAMAN PERSEMBAHAN

Bismillahirrahmanirrahim

Saya persembahkan karya tulis ini untuk Ibu, Ayah, Along, Angah dan si bungsu tercinta yang senantiasa mendo'akan dan mendukung saya untuk menyelesaikan karya tulis ini

Terimakasih kepada kedua pembimbing saya Bapak Heri Priyanto dan Bapak Novi Safriadi yang telah memberikan bimbingan dalam menyelesaikan karya tulis ini

Terimakasih kepada kedua penguji saya Bapak Helfi Nasution dan Bapak M.

Azhar Irwansyah yang telah memberikan saran dan masukan dalam menyelesaikan karya tulis ini

Terimakasih kepada seluruh Dosen dan Staff Informatika yang selalu memberikan teguran dan saran kepada saya

Terimakasih kepada seluruh teman-teman Teknik Informatika 2014 yang selalu memberikan semangat, teguran, tempat berbagi keluh kesah dan senda gurau yang telah berjuang dari awal masuk kuliah hingga saat ini

Terimakasih kepada abang-abang dan adik-adik aslab yang sudah memberikan dukungan sebagai penghuni Lab Informatika

Terimakasih kepada teman-teman yang berada diluar lingkungan kampus dan terimakasih kepada sanak saudara saya yang selalu memberikan semangat Dan untuk pihak-pihak lain yang tidak bisa saya sebutkan namanya satu persatu saya ucapkan **Jazakumullahu Khayran**

“Tidak ada sesuatu setelah kewajiban yang lebih afdhal daripada menuntut ilmu”

- Al-Imam Asy-Syafi'i -

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadiran Tuhan Yang Maha Esa, karena atas berkat dan rahmat yang diberikan-Nya, penulis bisa menyelesaikan skripsi yang berjudul “Implementasi *Infrastructure as a Service* pada *Cloud Computing* menggunakan *Load Balancing*” yang merupakan salah satu syarat untuk memperoleh gelar Sarjana Komputer pada Fakultas Teknik Universitas Tanjungpura Pontianak. Penelitian ini bertujuan untuk membangun dan menganalisis hasil performansi *load server* berbasis *cloud computing* pada Gedung Jurusan Informatika Untan.

Dalam penulisan ini penulis juga banyak menerima bimbingan dan bantuan dari berbagai pihak. Secara khusus penulis menyampaikan terima kasih yang sebesar-besarnya kepada Bapak Heri Priyanto, S.T., M.T. sebagai pembimbing utama dan Bapak Novi Safriadi, S.T., M.T. pembimbing pendamping yang telah memberikan bimbingan dan masukan dalam membangun aplikasi dalam penelitian ini. Terima kasih juga penulis sampaikan kepada Bapak Helfi Nasution, S.Kom., M.Cs. sebagai penguji utama dan Bapak M. Azhar Irwansyah, S.T., M.Eng. sebagai penguji pendamping yang telah memberikan masukan dan saran dalam penelitian ini.

Penulis berharap penelitian ini dapat bermanfaat bagi masyarakat sebagai media dalam menyampaikan informasi tentang abarsi pantai. Dalam penelitian dan penulisan skripsi ini penulis juga menyadari masih terdapat kekurangan, dan jauh dari kata sempurna sehingga masukan dan saran yang membangun masih diharapkan agar dapat menyempurnakan penelitian ini.

Pontianak, 2 Februari 2020

Agus Saputra

NIM. D1041141049

ABSTRAK

Pemanfaatan sumber daya komputasi yang lebih efisien dapat dilakukan dengan cara membangun sistem komputasi secara virtual pada *server* fisik. *Cloud computing* merupakan sumber daya seperti, *storage*, memori, CPU dan jaringan berbasis virtual yang memiliki banyak kelebihan dan mudah digunakan. Openstack merupakan salah satu *cloud computing* dengan model *infrastructure as a service* (IaaS) yang berbasis *opensource*. Virtualisasi yang dibangun dengan sumber daya standar pada *cloud computing* memiliki permasalahan pada pengguna *web server* yang semakin meningkat sehingga memerlukan *server* dengan sumber daya yang tinggi. Solusi yang dapat diimplementasikan untuk mengatasi masalah tersebut adalah dengan menciptakan sumber daya virtual dengan metode *load balancer*. Penelitian ini bertujuan untuk membangun dan menganalisa performa antara *load server*. Pengujian menggunakan metode *availability*, *quality of service* dan *workload*, serta pengujian dilakukan pada kondisi jaringan lokal, jaringan sepi dan jaringan sibuk. Hasil pengujian *availability* menunjukkan *web server* menggunakan algoritma *least connection* lebih unggul dari *web server* lain. Pada pengujian *quality of service* dan *workload* menunjukkan *web server* yang menggunakan algoritma *round robin* lebih unggul pada kondisi jaringan lokal dan sepi, sedangkan algoritma *least connections* lebih unggul pada kondisi jaringan sibuk. Berdasarkan pengujian yang telah dilakukan, *web server* algoritma *round robin* direkomendasikan untuk penerapan *web server* pada kualitas jaringan yang sepi dan *web server* tidak sering terjadi *down* dan algoritma *least connections* direkomendasikan untuk penerapan *web server* pada kualitas jaringan tidak stabil dan *web server* sering terjadi *down*.

Kata kunci: IaaS, *web server*, *availability*, *quality of service*, *workload*, *round robin*, *least connections*.

ABSTRACT

Utilization of more efficient computing resources can be done by building a virtual computing system on a physical server. Cloud computing is a resource like, storage, memory, CPU and virtual based network that has many advantages and easy to use. Openstack is one of cloud computing with an opensource-based infrastructure as a service (IaaS) model. Virtualization built with standard resources in cloud computing has an increasing problem in Web server users, requiring a high-resource Web server. A solution that can be implemented to address such problems is to create a virtual resource with a load Balancer method. This research aims to build and analyze the performance between server loads. Testing using availability, quality of service and workload methods, and testing is done on local network conditions, deserted networks and busy networks. The results of an availability test show that the Web server uses the least connection algorithms more excelling from other Web servers. On testing quality of service and workload shows the Web server using the round robin method is superior to stable network conditions and the least connections method is superior to unstable network conditions. Based on the tests that have been done, the Web server algorithms round robin is recommissioning for the application on stable network quality and the Web server does not often occur down and the least connections algorithm is recommended for implementation on Unstable network quality and Web server often occurs down.

Keywords: IaaS, Web server, availability, quality of service, workload, round robin, least connections.

DAFTAR ISI

HALAMAN PERNYATAAN.....	ii
HALAMAN PENGESAHAN.....	iii
HALAMAN PERSEMBAHAN.....	iv
KATA PENGANTAR.....	v
ABSTRAK.....	vi
DAFTAR ISI.....	viii
DAFTAR TABEL.....	xii
DAFTAR GAMBAR.....	xiii
DAFTAR LAMPIRAN.....	xv
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	2
1.3 Tujuan Penelitian.....	2
1.4 Batasan Masalah.....	3
1.5 Sistematika Penulisan.....	3
BAB II TINJAUAN PUSTAKA.....	4
2.1 Kajian Terkait.....	4
2.2 <i>Cloud Computing</i>	5
2.2.1 <i>Software as a Service</i>	5
2.2.2 <i>Platform as a Service</i>	6
2.2.3 <i>Infrastructure as a Service</i>	6
2.3 Openstack.....	7
2.4 <i>Load Balancing</i>	9
2.5 <i>Load Balancing as a Service</i>	10
2.5.1 <i>Round Robin</i>	13
2.5.2 <i>Least Connections</i>	14
2.5.3 <i>Source IP</i>	15
2.6 Pengujian Sistem.....	16
2.6.1 <i>Availability</i>	16
2.6.2 <i>Quality of Service</i>	16

2.6.3	<i>Workload</i>	18
2.7	Media Pengujian.....	18
2.8	<i>Tools</i> Pengujian.....	19
BAB III METODOLOGI PENELITIAN.....		20
3.1	Analisis Kebutuhan Sistem.....	20
3.1.1	Kebutuhan Perangkat Keras <i>Server</i>	20
3.1.2	Kebutuhan Perangkat Lunak <i>Server</i>	20
3.1.3	Kebutuhan Perangkat Keras Laptop.....	21
3.1.4	Kebutuhan Perangkat Lunak Laptop.....	21
3.1.5	Kebutuhan Perangkat Keras <i>Virtual Load Server</i>	21
3.1.6	Kebutuhan Perangkat Keras <i>Virtual Database Server</i>	21
3.1.7	Kebutuhan Perangkat Lunak <i>Virtual Server</i>	22
3.2	Metodologi Penelitian.....	22
3.3	Perancangan Sistem Openstack dan <i>Load Balancer</i>	24
3.4	Implementasi Sistem Openstack dan <i>Load Balancer</i>	26
3.4.1	Konfigurasi Jaringan Openstack.....	28
3.4.2	Konfigurasi <i>Load Balancer</i>	31
3.4.3	Konfigurasi <i>Instance Load Server</i>	32
3.5	Pengujian Sistem.....	32
3.5.1	Pengujian <i>Availability</i>	33
3.5.2	Pengujian <i>Quality of Service</i>	36
3.5.3	Pengujian <i>Workload</i>	39
3.6	Analisis Hasil Pengujian.....	39
3.7	Penarikan Kesimpulan.....	40
BAB IV IMPLEMENTASI DAN ANALISIS.....		41
4.1	Implementasi.....	41
4.2	Antarmuka Openstack.....	41
4.2.1	Antarmuka <i>Compute</i>	41
4.2.1.1	Antarmuka <i>Overview</i>	41
4.2.1.2	Antarmuka <i>Instance</i>	42
4.2.1.3	Antarmuka <i>Images</i>	42
4.2.1.4	Antarmuka <i>Key Pairs</i>	43
4.2.1.5	Antarmuka <i>Flavors</i>	44
4.2.2	Antarmuka <i>Volume</i>	44

4.2.2.1	Antarmuka <i>Volumes</i>	44
4.2.2.2	Antarmuka <i>Snapshots</i>	45
4.2.3	Antarmuka <i>Network</i>	45
4.2.3.1	Antarmuka <i>Network Topology</i>	46
4.2.3.2	Antarmuka <i>Networks</i>	46
4.2.3.3	Antarmuka <i>Routers</i>	47
4.2.3.4	Antarmuka <i>Security Group</i>	47
4.2.3.5	Antarmuka <i>Load Balancers</i>	48
4.2.3.6	Antarmuka <i>Floating IPs</i>	49
4.3	Hasil Pengujian <i>Availability</i>	49
4.3.1	Hasil Pengujian <i>Availability</i> Jaringan Lokal.....	50
4.3.2	Hasil Pengujian <i>Availability</i> Jaringan Sepi.....	52
4.3.3	Hasil Pengujian <i>Availability</i> Jaringan Sibuk.....	54
4.3.4	Analisis Hasil Pengujian <i>Availability</i>	56
4.4	Hasil Pengujian <i>Quality of Service</i>	57
4.4.1	Hasil Pengujian <i>Quality of Service</i> Jaringan Lokal.....	58
4.4.1.1	Pengujian <i>Throughput</i>	58
4.4.1.2	Pengujian <i>Delay</i>	60
4.4.1.3	Pengujian <i>Jitter</i>	62
4.4.1.4	Pengujian <i>Packet Loss</i>	63
4.4.2	Hasil Pengujian <i>Quality of Service</i> Jaringan Sepi.....	65
4.4.2.1	Pengujian <i>Throughput</i>	66
4.4.2.2	Pengujian <i>Delay</i>	67
4.4.2.3	Pengujian <i>Jitter</i>	69
4.4.2.4	Pengujian <i>Packet Loss</i>	71
4.4.3	Hasil Pengujian <i>Quality of Service</i> Jaringan Sibuk.....	73
4.4.3.1	Pengujian <i>Throughput</i>	73
4.4.3.2	Pengujian <i>Delay</i>	75
4.4.3.3	Pengujian <i>Jitter</i>	76
4.4.3.4	Pengujian <i>Packet Loss</i>	78
4.4.4	Analisis Hasil Pengujian <i>Quality of Service</i>	79
4.5	Hasil Pengujian <i>Workload</i>	81
4.5.1	Hasil Pengujian <i>Workload</i> Jaringan Lokal.....	81
4.5.2	Hasil Pengujian <i>Workload</i> Jaringan Sepi.....	83

4.5.3 Hasil Pengujian <i>Workload</i> Jaringan Sibuk.....	84
4.5.4 Analisis Hasil Pengujian <i>Workload</i>	86
4.6 Analisis Keseluruhan.....	86
BAB V PENUTUP.....	89
5.1 Kesimpulan.....	89
5.2 Saran.....	90
DAFTAR PUSTAKA.....	91

DAFTAR TABEL

Tabel 2.1 Kategori Latensi <i>Delay</i>	17
Tabel 2.2 Kategori Degradasi <i>Jitter</i>	18
Tabel 2.3 Kategori Degradasi <i>Packet Loss</i>	18
Tabel 3.1 Spesifikasi Jaringan Openstack.....	28
Tabel 3.2 Konfigurasi <i>Security Group</i>	30
Tabel 3.3 Hasil Pengujian <i>Downtime</i>	34
Tabel 3.4 Hasil Perhitungan <i>Availability</i>	35
Tabel 3.5 Hasil Perhitungan <i>Throughput</i>	36
Tabel 3.6 Hasil Perhitungan <i>Delay</i>	37
Tabel 3.7 Hasil Perhitungan <i>Jitter</i>	38
Tabel 3.8 Hasil Perhitungan <i>Packet Loss</i>	38
Tabel 3.9 Hasil Pengujian <i>Workload</i>	39
Tabel 4.1 Hasil Pengujian <i>Downtime</i> Jaringan Lokal.....	50
Tabel 4.2 Hasil Perhitungan <i>Availability</i> Jaringan Lokal.....	50
Tabel 4.3 Hasil Pengujian <i>Downtime</i> Jaringan Sepi.....	52
Tabel 4.4 Hasil Perhitungan <i>Availability</i> Jaringan Sepi.....	53
Tabel 4.5 Hasil Pengujian <i>Downtime</i> Jaringan Sibuk.....	54
Tabel 4.6 Hasil Perhitungan <i>Availability</i> Jaringan Sibuk.....	55
Tabel 4.7 Hasil Perhitungan <i>Throughput</i> Jaringan Lokal.....	58
Tabel 4.8 Hasil Perhitungan <i>Delay</i> Jaringan Lokal.....	60
Tabel 4.9 Hasil Perhitungan <i>Jitter</i> Jaringan Lokal.....	62
Tabel 4.10 Hasil Perhitungan <i>Packet Loss</i> Jaringan Lokal.....	64
Tabel 4.11 Hasil Perhitungan <i>Throughput</i> Jaringan Sepi.....	66
Tabel 4.12 Hasil Perhitungan <i>Delay</i> Jaringan Sepi.....	68
Tabel 4.13 Hasil Perhitungan <i>Jitter</i> Jaringan Sepi.....	70
Tabel 4.14 Hasil Perhitungan <i>Packet Loss</i> Jaringan Sepi.....	71
Tabel 4.15 Hasil Perhitungan <i>Throughput</i> Jaringan Sibuk.....	73
Tabel 4.16 Hasil Perhitungan <i>Delay</i> Jaringan Sibuk.....	75
Tabel 4.17 Hasil Perhitungan <i>Jitter</i> Jaringan Sibuk.....	77
Tabel 4.18 Hasil Perhitungan <i>Packet Loss</i> Jaringan Sibuk.....	78
Tabel 4.19 Hasil Pengujian <i>Workload</i> Jaringan Lokal.....	81
Tabel 4.20 Hasil Pengujian <i>Workload</i> Jaringan Sepi.....	83
Tabel 4.21 Hasil Pengujian <i>Workload</i> Jaringan Sibuk.....	85

DAFTAR GAMBAR

Gambar 2.1 Arsitektur openstack.....	7
Gambar 2.2 <i>Load balancer</i>	9
Gambar 2.3 <i>Load balancing as a service</i>	11
Gambar 2.4 <i>Load balancing as a service</i> dengan HAProxy.....	12
Gambar 2.5 Algoritma <i>round robin</i>	13
Gambar 2.6 Algoritma <i>least connection</i>	14
Gambar 3.1 Diagram alir penelitain.....	22
Gambar 3.2 Perancangan arsitektur jaringan.....	24
Gambar 3.3 Bandwith jaringan <i>wireless</i>	25
Gambar 3.4 Bandwith jaringan lokal.....	25
Gambar 3.5 Diagram alir instalasi openstack dan <i>load balancer</i>	26
Gambar 3.6 Membuat jaringan <i>public</i> dan <i>private</i>	29
Gambar 3.7 Membuat <i>virtual router</i>	30
Gambar 3.8 Membuat <i>security group</i>	31
Gambar 3.9 Membuat <i>load balancer</i>	31
Gambar 3.10 Konfigurasi <i>instance load server</i>	32
Gambar 3.11 Grafik penggunaan jaringan untan.....	33
 Gambar 4.1 Antarmuka <i>overview openstack</i>	42
Gambar 4.2 Antarmuka <i>instances</i>	42
Gambar 4.3 Antarmuka <i>images</i>	43
Gambar 4.4 Antarmuka <i>key pairs</i>	43
Gambar 4.5 Antarmuka <i>flavors</i>	44
Gambar 4.6 Antarmuka <i>volumes</i>	45
Gambar 4.7 Antarmuka <i>snapshots</i>	45
Gambar 4.8 Antarmuka <i>network topology</i>	46
Gambar 4.9 Antarmuka <i>networks</i>	47
Gambar 4.10 Antarmuka <i>routers</i>	47
Gambar 4.11 Antarmuka <i>security group</i>	48
Gambar 4.12 Antarmuka <i>load balancer</i>	48
Gambar 4.13 Antarmuka <i>floating ips</i>	49
Gambar 4.14 Hasil pengujian <i>availbility</i> jaringan lokal.....	51
Gambar 4.15 Hasil pengujian <i>availbility</i> jaringan sepi.....	53
Gambar 4.16 Hasil pengujian <i>availbility</i> jaringan sibuk.....	55
Gambar 4.17 Hasil pengujian <i>throughput</i> jaringan lokal.....	59
Gambar 4.18 Hasil pengujian <i>delay</i> jaringan lokal.....	61
Gambar 4.19 Hasil pengujian <i>jitter</i> jaringan lokal.....	63
Gambar 4.20 Hasil pengujian <i>packet loss</i> jaringan lokal.....	65
Gambar 4.21 Hasil pengujian <i>throughput</i> jaringan sepi.....	67
Gambar 4.22 Hasil pengujian <i>delay</i> jaringan sepi.....	69

Gambar 4.23 Hasil pengujian <i>jitter</i> jaringan sepi.....	70
Gambar 4.24 Hasil pengujian <i>packet loss</i> jaringan sepi.....	72
Gambar 4.25 Hasil pengujian <i>throughput</i> jaringan sibuk.....	74
Gambar 4.26 Hasil pengujian <i>delay</i> jaringan sibuk.....	76
Gambar 4.27 Hasil pengujian <i>jitter</i> jaringan sibuk.....	77
Gambar 4.28 Hasil pengujian <i>packet loss</i> jaringan sibuk.....	79
Gambar 4.29 Hasil pengujian <i>workload</i> jaringan lokal.....	82
Gambar 4.30 Hasil pengujian <i>workload</i> jaringan sepi.....	84
Gambar 4.31 Hasil pengujian <i>workload</i> jaringan sibuk.....	85

DAFTAR LAMPIRAN

Lampiran A-1 Pengujian <i>Availability</i> Jaringan Lokal.....	A-1
Lampiran A-2 Pengujian <i>Availability</i> Jaringan Sepi.....	A-3
Lampiran A-3 Pengujian <i>Availability</i> Jaringan Sibuk.....	A-6
Lampiran B-1 Pengujian <i>Quality of Service</i> Jaringan Lokal.....	B-1
Lampiran B-2 Pengujian <i>Quality of Service</i> Jaringan Sepi.....	B-11
Lampiran B-3 Pengujian <i>Quality of Service</i> Jaringan Sibuk.....	B-22
Lampiran C-1 Pengujian <i>Workload</i> Jaringan Lokal.....	C-1
Lampiran C-2 Pengujian <i>Workload</i> Jaringan Sepi.....	C-5
Lampiran C-3 Pengujian <i>Workload</i> Jaringan Sibuk.....	C-9
Lampiran D-1 Konfigurasi Jaringan <i>Bare-metal Server</i>	D-1
Lampiran D-2 <i>Software Repositories</i>	D-1
Lampiran D-3 Instal Openstack.....	D-1
Lampiran D-4 Konfigurasi <i>Load Balancer</i>	D-1
Lampiran D-5 <i>Deploy Load Balancer</i>	D-1
Lampiran D-6 Instal <i>Dashboard Load Balancer</i>	D-2
Lampiran D-7 Aktifkan <i>Load Balancer</i> di <i>Dashboard</i>	D-2
Lampiran D-8 <i>Restart Apache</i>	D-2

BAB I

PENDAHULUAN

1.1 Latar Belakang

Seiring perkembangan teknologi yang terus berkembang dan mempengaruhi perkembangan komputasi tradisional menjadi komputasi awan (*cloud*). *Cloud computing* merupakan komputasi dimana semua *resource* dan sumber daya komputer baik itu memori, aplikasi, prosesor, jaringan dan sistem operasi, yang digunakan secara *virtual* dengan pola akses remote sehingga klien bisa mengakses layanan tersebut kapanpun dan dimanapun selama terhubung dengan jaringan internet.

Cloud computing memiliki tiga tingkatan layanan utama yang diberikan kepada pengguna, yaitu *software as a service*, *platform as a service*, *infrastructure as a service*. Beberapa perangkat lunak *opensource* yang digunakan untuk merancang *public*, *private* atau *hybrid cloud computing*, salah satunya seperti *openstack*. *Openstack* adalah sebuah *opensource software platform* yang digunakan untuk pengembangan layanan *infrastructure as a service* pada *cloud computing*. *OpenStack* digunakan untuk membangun infrastruktur komputasi jaringan karena sifatnya yang *open source* dan bisa ditambahkan fitur *load balancer* sehingga lebih efisien dan performansinya yang lebih baik dibandingkan platform *cloud computing* lainnya.

Load balancer adalah teknik untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal, memaksimalkan *throughput*, memperkecil waktu tanggap dan menghindari overload pada salah satu jalur koneksi. *Load balancer* digunakan pada saat sebuah *server* telah memiliki jumlah *request* yang telah melebihi maksimal kapasitasnya. *Load balancing* juga mendistribusikan beban kerja secara merata di dua atau lebih komputer, link jaringan, *cpu*, *hard drive*, atau sumber daya lainnya, untuk mendapatkan pemanfaatan sumber daya yang optimal. Salah satu jenis *load balancer* adalah *HaProxy*, memiliki kemampuan mengontrol trafik dari masing-masing *request* data dari klien, bukan hanya berdasarkan jumlah koneksi yang

masuk, selain itu HaProxy dipilih karena terdapat fitur untuk menampilkan statistik dari penggunaan masing-masing *web server* yang dikontrolnya. Untuk mengoptimalkan kinerja *load balancer* maka perlu menggunakan algoritma atau metode untuk mengatur penjadwalan proses pembagian koneksi.

Seiring dengan bertambahnya pengguna *web server* pada suatu *server* sehingga membuat kinerja *web server* bertambah berat. *Web server* yang baik tentunya mampu melayani *request* dalam jumlah yang besar pada satu waktu. Seperti halnya pada Gedung Jurusan Informatika yang memiliki *server* secara *cloud computing* dan *virtual server* yang dibangun dalam kondisi tunggal yang memiliki spesifikasi standar sehingga tidak mampu diakses oleh klien dalam jumlah yang banyak dalam satu waktu. *Virtual server* tersebut memiliki *web server* yang dapat diakses oleh klien yang sewaktu-waktu bisa mengalami *down*. Hal ini tentu akan mengganggu proses pertukaran data yang terjadi antara *web server* dengan klien. Salah satu solusi yang bisa dilakukan adalah dengan cara meng-*upgrade* spesifikasi *server* dengan biaya yang cukup besar. Salah satu solusi yang dianggap tepat untuk mengatasi masalah tersebut adalah dengan menggunakan metode *load balancing*.

Oleh karena itu, untuk mengatasi masalah yang ada, pada penelitian yang akan dilakukan yaitu bagaimana cara membangun sebuah *server* berbasis *cloud computing* dengan metode *load balancing* yang dapat bekerja secara optimal sehingga dapat memenuhi kebutuhan pengguna.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, maka permasalahan yang dapat diambil adalah bagaimana membangun dan menganalisa kinerja *load server* pada *cloud computing* dan merekomendasikan metode *load balancing* yang tepat untuk diterapkan pada Gedung Jurusan Informatika Untan.

1.3 Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk membangun dan menganalisis hasil performansi *load server* berbasis *cloud computing* pada Gedung Jurusan Informatika Untan.

1.4 Batasan Masalah

Pembatasan masalah dari penelitian yang akan dilakukan antara lain sebagai berikut:

1. Menggunakan 1 unit *server* sebagai *cloud server* dan 1 unit laptop sebagai pengujian.
2. *Platform* yang digunakan untuk membangun *infrastructure as a service* adalah *openstack*.
3. Menggunakan algoritma *round robin*, *least connection* dan *source ip* sebagai implementasi metode *load balancing*.
4. Menggunakan metode pengujian *availability*, *quality of service* dan *workload*
5. Tidak membahas keamanan jaringan.
6. Penelitian dilakukan pada Gedung Jurusan Informatika Untan.

1.5 Sistematika Penulisan

Adapun sistematika penulisan dari tugas akhir ini disusun dalam lima bab yang terdiri dari Bab I Pendahuluan, Bab II Tinjauan Pustaka, Bab III Metodologi Penelitian, Bab IV Implementasi dan Pengujian serta Bab V Penutup.

Bab I Pendahuluan adalah bab yang berisi latar belakang, perumusan masalah, tujuan penelitian, pembatasan masalah dan sistematika penulisan.

Bab II Tinjauan Pustaka tentang hasil-hasil penelitian yang didapat oleh peneliti terdahulu dan berisi literatur atau landasan teori yang digunakan dalam penelitian.

Bab III Metodologi Penelitian adalah bab yang berisi tentang data penelitian, alat yang dipergunakan, metode penelitian, analisis hasil serta diagram alir penelitian.

Bab IV Implementasi dan Pengujian adalah bab yang berisi data hasil percobaan, penerapan, pengujian dan sebagainya yang telah dirancang pada Bab III. Setiap hasil yang disajikan akan dilakukan analisis untuk mengarah kepada suatu kesimpulan.

Bab V Penutup adalah bab yang berisi kesimpulan dari penelitian yang telah dilakukan serta saran dan rekomendasi untuk perbaikan, pengembangan atau kesempurnaan/kelengkapan penelitian yang telah dilakukan.

BAB II

TINJAUAN PUSTAKA

2.1 Kajian Terkait

Prayudi Aditya Nugraha (2016) melakukan penelitian Rancang Bangun *Web Server* Berbasis Linux dengan Metode *Load Balancing* pada Gedung Jurusan Informatika Universitas Tanjungpura. Tujuan penelitian ini untuk menganalisis algoritma penjadwalan *load balancing* dan meningkatkan ketersediaan *web server*. Pengujian pada penelitian ini dilakukan dengan cara mengakses *web server* dalam keadaan *down* untuk mendapatkan hasil ketersediaan layanan, *throughput* dan waktu *respon*. Hasil pengujian menunjukkan bahwa *web server* dengan algoritma *load balancing* dapat memberikan ketersediaan *web server* lebih baik dibandingkan *web server* tunggal. Pengujian *throughput* pada *web server load balancing* menggunakan algoritma round robin memiliki nilai yang paling baik yaitu sebesar 0,19 MB/detik, sedangkan *web server* yang menggunakan algoritma *least connections* memiliki nilai *throughput* paling kecil yaitu sebesar 0,17 MB/detik. Pengujian waktu respon pada *web server load balancer* menggunakan algoritma *least connections* memiliki waktu respon tercepat yaitu 0,258 detik, sedangkan *web server* tunggal memiliki waktu respon terlama yaitu 0,24 detik.

Hari Triyanto (2019) melakukan penelitian Analisa Perbandingan Performa Openstack dan Apache Cloudstack dalam Model Cloud Computing Berbasis *Infrastructure as a Service*. Penelitian ini bertujuan untuk menganalisa perbandingan performa openstack dan cloudstack dalam model *cloud computing* berbasis *infrastructure as a service*. Penelitian ini menggunakan metode pengujian *overhead* dan *linearity* untuk mendapatkan perbandingan dari aspek pengujian *web server*, komputasi, *database* dan jaringan. Hasil pengujian menunjukkan bahwa openstack unggul dalam aspek komputasi dan *web server*, sehingga direkomendasikan untuk penggunaan aplikasi berbasis *web* atau aplikasi yang memerlukan daya komputasi tinggi. Sedangkan cloudstack unggul dalam

aspek database dan jaringan, sehingga direkomendasikan untuk penggunaan aplikasi yang memerlukan penyimpanan yang besar.

Yulizar Pribadi (2019) melakukan penelitian Analisis Penggunaan Metode *Failover Clustering* untuk Mencapai *High Availability* pada *web server* pada Gedung Jurusan Informatika Universitas Tanjungpura. Tujuan penelitian ini untuk melihat penggunaan metode *failover clustering* dalam mencapai layanan *high availability* pada *web server*. Penelitian ini menggunakan metode pengujian *availability*, *workload* dan *quality of service* untuk membandingkan peforma dari *web server* yang menggunakan metode *failover clustering* dan *web server* tanpa *failover clustering*. Hasil pengujian menunjukkan bahwa *web server failover clustering* masih bisa diakses meskipun *web server* utama terjadi *down*. Hasil pengujian *availability* dapat memberikan ketersediaan layanan yang lebih baik saat terjadi kegagalan dan memiliki nilai *availability* sebesar 99,90%. Hasil pengujian *workload* pada *web server* tanpa *failover* lebih tinggi yaitu sebesar 808 *request*, sedangkan pada *web server failover* lebih rendah yaitu 806 *request*, hal ini dikarenakan *web server failover* membutuhkan *resource* yang besar. Hasil pengujian *quality of service* pada *web server failover* dan tanpa *failover* menunjukkan nilai indeks yang memuaskan dan peforma *web server failover* lebih baik dari *web server* tanpa *failover* dari parameter *quality of service*.

2.2 Cloud Computing

Cloud Computing adalah sebuah mekanisme yang memungkinkan kita "menyewa" sumber daya teknologi informasi berupa *software*, *processing power*, *storage*, dan lainnya, melalui internet dan memanfaatkan sesuai kebutuhan kita dan membayar sesuai dengan yang digunakan. Dengan konsep ini, maka semakin banyak orang yang bisa memiliki akses dan memanfaatkan sumber daya tersebut, karena tidak harus melakukan *investasi* besar-besaran (Anggeriana, 2011). Ada 3 jenis layanan pada *cloud computing* yang diberikan kepada *user*, yaitu : *software as a service*, *platform as a service*, *infrastructure as a service*.

2.2.1 Software as a Service

Software as a Service merupakan layanan yang diberikan dengan menyediakan *software* maupun aplikasi yang dapat diakses pelanggan dari internet (Fajrin, 2012). Penyedia layanan *cloud computing* berinteraksi dengan

pengguna dan pelanggan melalui sebuah *front-end panel*. *Software as a service* menghapus kebutuhan organisasi untuk melakukan instalasi dan menjalankan aplikasi pada komputer atau data center mereka. Pengguna dapat menggunakan aplikasi tanpa harus mengerti bagaimana data disimpan, melakukan perawatan dan pengembangan aplikasi tersebut. Keuntungan dari *software as a service* adalah pengguna hanya berfokus pada layanan yang akan digunakan. Seluruh proses pengadaan seperti perancangan, instalasi dan perawatan server maupun aplikasi ditanggung oleh penyedia layanan. Contoh dari *software as a service* adalah Microsoft Office 365, Google Docs dan Dropbox.

2.2.2 Platform as a Service

Platform as a Service merupakan layanan yang diberikan kepada konsumen untuk menyebarkan aplikasi yang dibuat konsumen atau diperoleh ke infrastruktur *cloud computing* menggunakan bahasa pemrograman dan peralatan yang didukung oleh *provider* (Ashari & Setiawan, 2011). *Platform as a service* menjadi wadah bagi pengguna untuk menjalankan aplikasinya. Penyediaan platform bagi developer yang dikelola oleh pihak *provider* dan diakses melalui internet. *Platform as a service* memungkinkan kita untuk membangun aplikasi, *upload* aplikasi, melakukan testing aplikasi, ataupun mengatur konfigurasi yang dibutuhkan dalam proses pengembangan aplikasi. Pengguna hanya perlu fokus pada pengembangan aplikasi karena *maintenance* sistem dikelola oleh pihak *provider*.

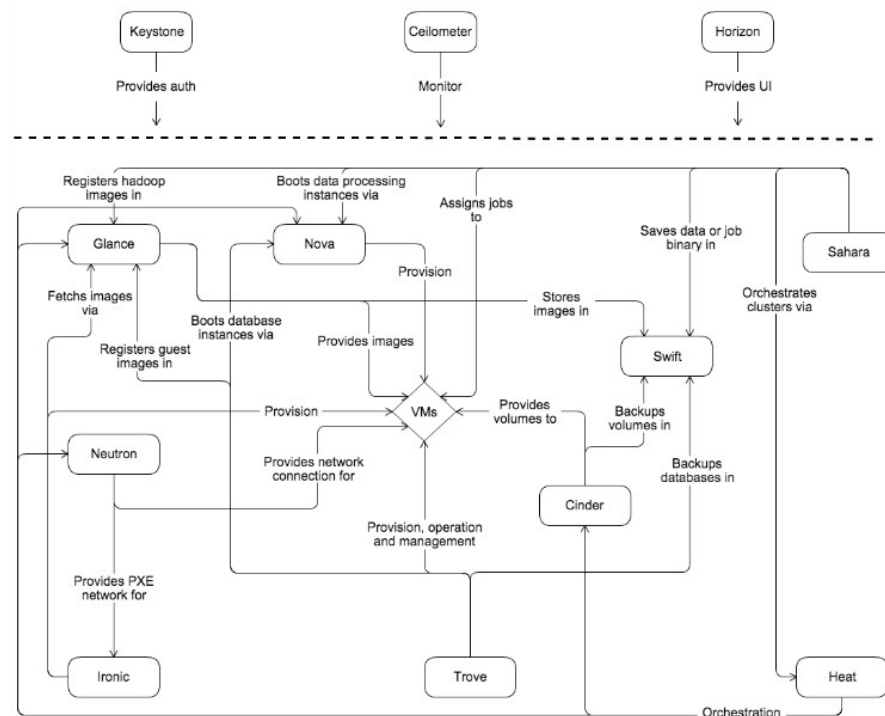
2.2.3 Infrastructure as a Service

Infrastructure as a service merupakan kemampuan yang diberikan kepada pengguna untuk memproses, menyimpan, mengkonfigurasi jaringan, dan sumber komputasi penting yang lain, dimana konsumen dapat menyebarkan dan menjalankan perangkat lunak secara bebas, yang dapat mencakup sistem operasi aplikasi (Ashari & Setiawan, 2011). *Infrastructure as a service* menyediakan sumber daya komputer yang tervirtualisasi dan dapat diakses melalui internet. Sumber daya tersebut dipesan oleh pengguna sesuai dengan kebutuhan dan biaya yang dimiliki oleh pengguna. Pengguna dapat memilih

sistem dengan berbagai varian cpu, memori dan penyimpanan. Selain itu pengguna juga dapat melakukan konfigurasi jaringan meskipun bersifat terbatas pada kebijakan *provider*. *Infrastructure as a service* memungkinkan pengguna dalam memilih dan mengkonfigurasi sistem operasi yang disediakan oleh *provider*. Pengguna dapat mengakses terminal dan melakukan instalasi *package* untuk mendukung pengembangan aplikasi. Contoh dari *infrastructure as a service* dengan model penyebaran *public* adalah Google Compute Engine, Amazon Elastic Cloud dan Microsoft Azure. Untuk model penyebaran *private*, *infrastructure as a service* dapat dibangun menggunakan *openstack* dengan sistem operasi Ubuntu.

2.3 Openstack

Openstack adalah *opensource cloud computing software* untuk membangun infrastruktur *cloud* yang *reliable* (Anggeriana, 2011). Openstack merupakan sebuah *opensource software platform* yang digunakan untuk pengembangan layanan *infrastructure as a service* pada *cloud computing*. Openstack dirancang dengan tiga komponen layanan utama yaitu *networking* sebagai jaringan pada *cloud*, *compute* sebagai komputasi pada *cloud* dan *storage* sebagai media penyimpanan pada *cloud* yang dapat digunakan sebagai media virtualisasi *server*. Arsitektur *openstack* merujuk pada Gambar 2.1.



Gambar 2.1 Arsitektur openstack (sumber : www.openstack.org)

Komponen-komponen yang ada pada openstack memiliki fungsi masing-masing, diantaranya sebagai berikut :

1. *Networking (Neutron)*

OpenStack Networking atau *neutron* merupakan sistem yang melakukan *provisioning* jaringan seperti mengatur jaringan atau *subnet*, *router*, *load balancing*, *gateway* dan *floating ip* yang melibatkan entitas pada *virtual machine*. *Neutron* juga berfungsi sebagai penyedia *network as a service* pada *cloud computing*.

2. *Compute (Nova)*

Openstack sompute adalah otak dari *cloud* dan dapat mengelola jaringan dengan skala besar secara *virtual*. *Nova* merupakan bagian utama dari sistem IaaS yang memungkinkan *user* untuk membuat dan mengelola *server* secara *virtual*. *Nova* juga dapat mengatur fungsi proses dan alokasi CPU untuk setiap *virtual machine*.

3. *Block Stroge (Cinder)*

Cinder menyediakan layanan penyimpanan blok untuk digunakan oleh *compute Instance* (klien). *Cinder* memungkinkan *user* untuk mengatur kebutuhan media penyimpanan dan dapat digunakan untuk penyimpanan *database*,

expandable file sistem, akses pada penyimpanan blok, *snapshot management*. Contohnya backup atau restorasi.

4. *Image Service (Glance)*

Glance adalah jenis komponen dari openstack *Image* yang menyediakan dan mengelola layanan Gambar serta bertanggung jawab dalam penyimpanan Gambar untuk Openstack.

5. *Identity (Keystone)*

Keystone menyediakan *service* identitas untuk openstack, bertindak sebagai sistem otentikasi standar seperti *username* dan *password* melalui sistem operasi di *cloud*.

6. *Object Storage (Swift)*

Swift termasuk kedalam jenis komponen openstack *object storage*. *Swift* menyediakan tempat penyimpanan untuk menyimpan objek, *file* atau data dalam jumlah besar. *Swift* dapat menyimpan dan menerima data yang sangat banyak dan dapat di-scale dengan mudah. *Swift* ideal untuk menyimpan data tidak terstruktur yang dapat berkembang tanpa batas.

7. *Orchestration (Heat)*

Heat dapat digunakan sebagai layanan untuk mengatur beberapa aplikasi di atas *cloud* misal menggunakan *REST API*. *Heat* ini biasa digunakan untuk mengatur lebih banyak *server* dengan cara melakukan otomatis pada layer *orchestration*.

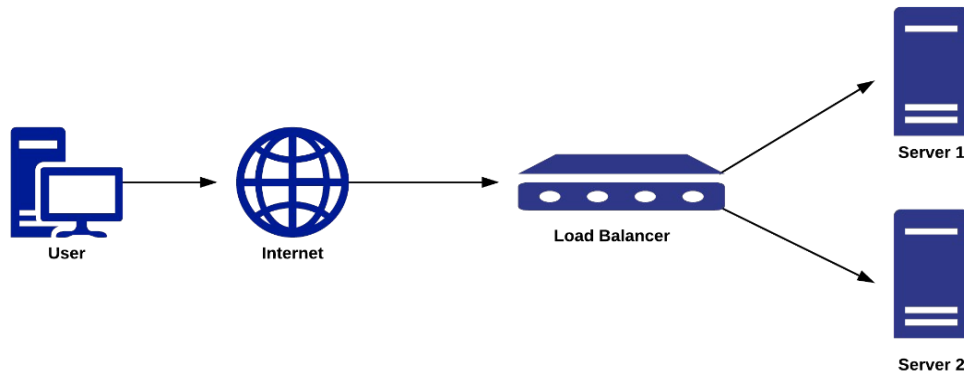
8. *Dashboard (Horizon)*

Horizon merupakan tampilan *dashboard* untuk *Openstack*. *Horizon* menyediakan antarmuka *web* untuk semua komponen dalam *Openstack*.

2.4 ***Load Balancing***

Load balancing adalah teknik untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal, memaksimalkan *throughput*, memperkecil *response time* dan menghindari *overload* pada salah satu jalur koneksi (Ramadhan, Latuconsina, & Purboyo, 2019). *Load balancing* digunakan pada saat sebuah *server* memiliki jumlah *request* yang telah melebihi kapasitas maksimal. *Load balancing* juga

mendistribusikan beban kerja secara merata di dua atau lebih komputer, *link* jaringan, CPU, *hard drive*, atau sumber daya lainnya, untuk mendapatkan pemanfaatan sumber daya yang optimal. *Load balancing* merujuk pada Gambar 2.2.



Gambar 2.2 *Load balancer* (analisa dari sumber : <https://cdn.keycdn.com>)

Ketika mengalami masalah dimana *server* telah mencapai batasnya, hal yang pertama kali dilakukan oleh seorang sistem *administrator* untuk menanganinya adalah dengan menambahkan RAM, ataupun prosesor pada *server* tersebut. Hal tersebut tidak bisa dilakukan terus menerus dikarenakan keterbatasan fisik dari suatu *server*. Oleh karena itu dibutuhkan suatu sistem *load balancing* dimana semua *request* yang datang tidak dibebankan oleh satu *server* tunggal.

Server yang menggunakan *load balancer* memiliki beberapa keuntungan, yang memiliki tingkat kesuksesan yang sangat tinggi dan banyak diterapkan pada teknologi saat ini. Beberapa keuntungan tersebut adalah:

1. *Flexibility*

Load balancing mengizinkan untuk menambah ataupun menghilangkan *server* dari daftar *server* yang berada pada *back-end* setiap saat. Hal ini memungkinkan untuk melakukan *maintenance* pada *server* manapun, bahkan pada saat jam sibuk sekalipun tanpa berdampak pada layanan suatu *server*.

2. *High Availability*

Load balancer juga dapat melakukan pemeriksaan pada status *server* yang tersedia, menghilangkan *server* dari daftar apabila *server* tersebut mengalami *down*, dan mengembalikan *server* ke dalam daftar jika *server* tersebut telah kembali aktif. Proses ini berjalan secara otomatis tanpa bantuan dari seorang

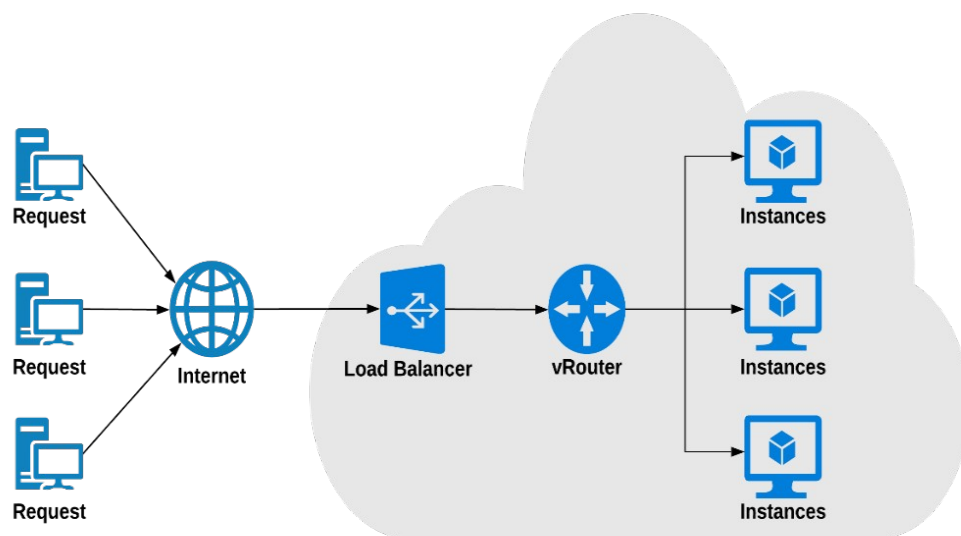
sistem *administrator*. Dan juga *load balancer* itu sendiri dapat dikonfigurasi untuk redundansi, apabila salah satu dari *load balancer* mengalami kegagalan.

3. Scalability

Karena *load balancer* bertugas untuk melakukan pendistribusian trafik, yang perlu dilakukan untuk membangun suatu layanan yang kuat adalah dengan cara menambahkan jumlah dari *server*. Hal ini sangat membantu dikarenakan lebih banyak membantu dari segi ekonomi karena beberapa *low-end server* lebih murah daripada suatu *high-end server*. Dan ketika beban meningkat, *server* tersebut dapat langsung digunakan secara bersamaan untuk menangani peningkatan trafik.

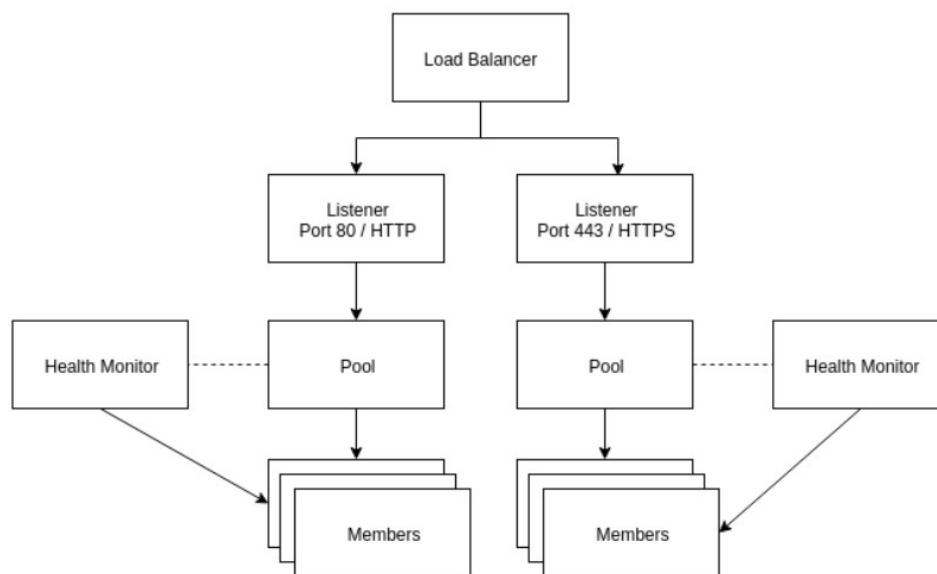
2.5 Load Balancing as a Service

Load balancing as a Service merupakan salah satu layanan yang ada pada komponen *node network* yaitu *neutron* pada *openstack*, dimana layanan ini dikembangkan berdasarkan riset yang dilakukan oleh komunitas maupun pengembang *openstack* (Adrika, Perdana, & Sanjoyo, 2018). *Load balancing* adalah teknologi yang digunakan untuk menciptakan *high availability* pada sumber daya komputasi melalui jaringan. Konsep *Load balancing* yaitu mendistribusikan jumlah *request* secara merata dengan membagi beban pada dua atau lebih jalur koneksi melalui beberapa *node* atau *server*. *Load balancing as a service* merujuk pada Gambar 2.3.



Gambar 2.3 *Load balancing as a service* (analisa dari sumber : <https://blog.rackspace.com>)

Beban dapat berupa beban jaringan, memori, CPU dan lain lain. *Load balancing* dapat diimplementasikan untuk *website* agar dapat meningkatkan kecepatan akses dari *website* saat dibuka. Selain itu juga redundansi dari *website* dengan *Load balancing* ini akan rendah. Banyak aplikasi lain yang dapat megimplementasikan teknologi *Load balancing* ini. *Load balancing as a service* merupakan salah satu layanan dari *openstack* yang memanfaatkan HAProxy. HAProxy merupakan salah satu *opensource* yang menyediakan *load balancer* berupa *software-based* dan merupakan mesin penyeimbang beban bawaan yang ada di *openstack*. *Load balancing as a service* dengan HAProxy merujuk pada Gambar 2.4.



Gambar 2.4 *Load balancing as a service* dengan HAProxy (sumber: <https://openstack.org>)

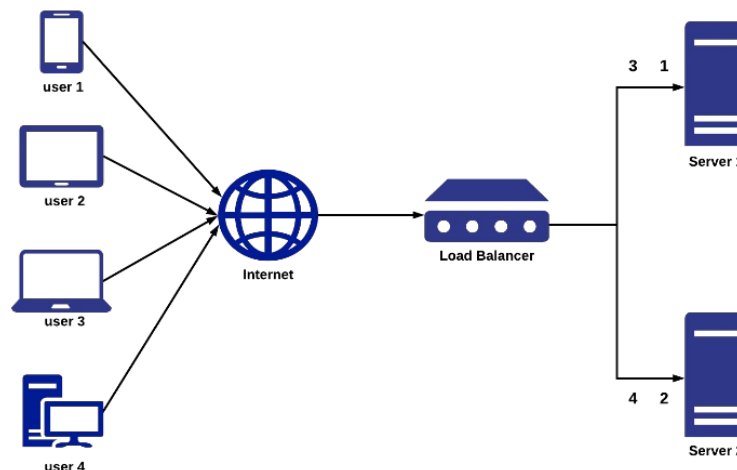
Penyeimbang beban berbasis HAProxy ini memiliki akses jaringan ke *client* dengan mengirim dan menerima *request* pada *neutron* menggunakan alamat IP yang menggunakan *virtual IP* dalam jaringan *openstack*. *Load balancer* adalah penyeimbang beban yang menempati *port* jaringan *neutron* dan memiliki alamat IP yang ditetapkan dari *subnet*. *Listener* merupakan *Load balancer* yang dapat

berfungsi mendengarkan permintaan pada beberapa *port* contohnya *port* 80 untuk HTTP dan *port* 443 untuk HTTPS. Masing-masing *port* tersebut ditentukan oleh *Listener*. *Pool* sebagai manajemen daftar anggota yang menyajikan konten melalui penyeimbang beban.

Members atau anggota adalah *server* yang melayani lalu lintas di belakang penyeimbang beban. Setiap anggota ditentukan oleh alamat IP dan *port* yang digunakannya untuk melayani lalu lintas. Anggota dapat mengalami offline dari waktu ke waktu dan *Health Monitor* mengalihkan lalu lintas dari anggota yang tidak merespons dengan benar. *Health monitor* terhubung dengan *pool*. Ada beberapa macam algoritma *load balancing*.

2.5.1 Round Robin

Round Robin adalah algoritma yang paling umum digunakan dalam *load balancer* pada *cloud computing*, merupakan algoritma dengan metode sederhana dan mudah untuk diterapkan (Ramadhan, Latuconsina, & Purboyo, 2019). Algoritma penjadwalan *round robin* akan memperlakukan semua *server* dengan sama tanpa memperhatikan jumlah koneksi ataupun *response time*. Algoritma *round robin* merujuk pada Gambar 2.5.



Gambar 2.5 Algoritma *round robin* (analisa dari sumber : <https://www.citrix.com>)

Kelebihan algoritma *round robin*, yaitu :

1. Mudah untuk diimplementasikan.
2. Semua proses memiliki kepentingan yang sama, sehingga tidak ada prioritas.
3. *Respond time* lebih cepat untuk proses yang berukuran kecil.
4. Dapat menghindari ketidakadilan layanan terhadap proses yang kecil.

Kekurangan algoritma *round robin*, yaitu :

1. Menentukan besarnya *time quantum*.
2. Dapat terjadi *overload* jika ukuran suatu *server* melebihi batas kemampuan *server* itu sendiri.

Berikut ini merupakan contoh algoritma *round robin* :

```

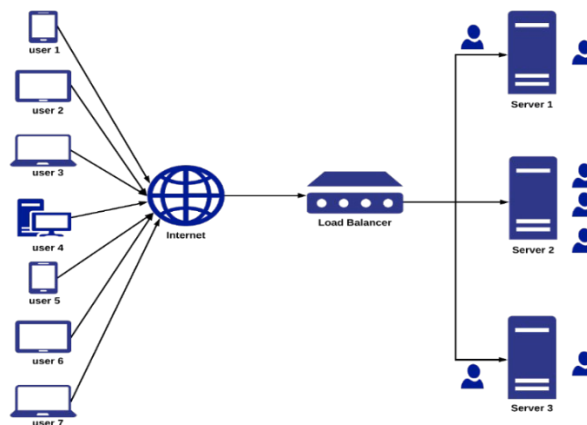
Supposing that there are n servers in the set  $S = \{S_0, S_1, \dots, S_{n-1}\}$ , where  $n > 0$ ;
i indicates the server selected last time, and i is initialized with -1;

j = i;
do {
    j = (j + 1) mod n;
    if (Available( $S_j$ )) {
        i = j;
        return  $S_i$ ;
    }
} while (j != i);
return NULL;

```

2.5.2 Least Connections

Least connection merupakan algoritma penjadwalan yang mengarahkan koneksi pada sebuah jaringan kepada *server* dengan melihat *server* yang memiliki jumlah koneksi aktif paling sedikit (**Adenan, Abdurohman, & Jadied, 2018**). Pada *virtual server* terdapat sekumpulan *server* dengan kinerja yang mirip, penjadwalan *least connection* sangat baik untuk melancarkan pendistribusian *load* ketika beban *request* sangat bervariasi dalam jumlah banyak. Hal ini karena semua pemrosesan *request* yang panjang memiliki kesempatan yang sangat kecil untuk diarahkan pada *server*. Algoritma *least connection* merujuk pada Gambar 2.6.



Gambar 2.6 Algoritma *least connection* (analisa dari sumber :
<https://www.citrix.com>)

Jika ada dua atau lebih *server* memiliki jumlah koneksi aktif yang sama, ada beberapa cara untuk memutuskan *server* mana yang akan digunakan, yaitu memilih *IP address* terkecil.

Kelebihan algoritma *least connection*, yaitu;

1. Menghindari terjadinya *overload* pada *server*.
2. Sangat cocok untuk diimplementasikan jika beban *request* yang bervariasi dalam jumlah yang banyak.

Kekurangan algoritma *least connection*, yaitu:

1. Perlu adanya konfigurasi tambahan pada *load balancer*.
2. Beban kerja *server* jadi tidak merata.

Berikut ini merupakan contoh algoritma *least connections* :

```

Supposing that there is a server set  $S = \{S_0, S_1, \dots, S_{n-1}\}$ ,
 $W(S_i)$  is the weight of server  $S_i$ ,
 $C(S_i)$  is the current connection number of server  $S_i$ ,

for ( $m = 0$ ;  $m < n$ ;  $m++$ ) {
    if ( $W(S_m) > 0$ ) {
        for ( $i = m+1$ ;  $i < n$ ;  $i++$ ) {
            if ( $W(S_i) \leq 0$ )
                continue;
            if ( $C(S_i) < C(S_m)$ )
                 $m = i$ ;
        }
        return  $S_m$ ;
    }
}
return NULL;

```

2.5.3 Source IP

Source IP adalah algoritma penjadwalan dengan cara mengatur permintaan dari alamat IP unik secara konsisten untuk diarahkan ke *server* yang sama. Pada algoritma *source ip*, untuk memilih *server* mana yang akan mengirim permintaan didasarkan pada alamat IP sumber sebagai kunci. Jika *server* tersedia,

permintaan akan dikirim ke *server*, atau akan dikembalikan bahwa *server* tidak dapat diakses (Cakrawardaya, Mayasari, & Sanjoyo, 2017).

2.6 Pengujian Sistem

Pengujian yang akan dilakukan pada penelitian ini adalah untuk menganalisa hasil performansi dari *load balancer* yang diimplementasikan pada openstack. Pengujian ini dilakukan untuk mendapatkan hasil pengukuran dari metode *availability*, *quality of service* dan *workload*.

2.6.1 Availability

availability adalah suatu kemampuan dari suatu sistem untuk melakukan fungsinya secara berkesinambungan (tanpa adanya interupsi) untuk jangka waktu lebih lama dari pada ketahanan yang di berikan oleh masing-masing komponennya (Umam, Handoko, & Rizqi, 2018). Secara garis besar *availability* merupakan kemampuan pada suatu layanan yang tersedia mampu memberikan layanan yang baik dalam waktu tertentu.

2.6.2 Quality of Service

Menurut Yanto (2013), *Quality of Service* (QoS) adalah kemampuan suatu jaringan untuk menyediakan layanan yang baik dengan menyediakan *bandwidth*, mengatasi *jitter* dan *delay*. *Quality of service* digunakan untuk mendapatkan informasi jaringan yang digunakan oleh *user*. *Quality of service* yang baik memiliki nilai *delay* dan *jitter* yang relative rendah sedangkan nilai *bandwidth* yang relatif tinggi. Nilai *quality of service* sangat dipengaruhi oleh perangkat keras yang digunakan seperti *router*, *switch* dan *access point*.

Pada penerapan *cloud computing*, *user* dapat mengatur jaringan secara virtual pada *virtual router*. *Quality of service* dapat digunakan untuk menentukan kualitas layanan jaringan dari *client* yang terhubung ke *instance* atau *virtual server*. Adapun parameter yang digunakan untuk menentukan nilai *Quality of Service* adalah sebagai berikut.

1. *Throughput*

Menurut Sasmita (Sasmita, safriadi, dan Irwansyah, 2013), *throughput* merupakan suatu parameter yang menggambarkan suatu kondisi yang menunjukkan jumlah total paket yang hilang, dapat terjadi karena *collision* dan *congestion* pada jaringan dan hal ini berpengaruh pada semua aplikasi karena retransmisi akan mengurangi efisiensi jaringan secara keseluruhan meskipun jumlah *bandwidth* cukup tersedia untuk aplikasi-aplikasi tersebut. *Throughput* merupakan total jumlah paket data yang sukses diterima selama interval waktu tertentu dan dibagi oleh durasi interval waktu tersebut. *Throughput* tidak sama dengan *bandwidth*. *Throughput* merupakan kecepatan aktual dari total *bandwidth* jaringan dari *client* ke sumber tujuan, sedangkan *bandwidth* merupakan kecepatan maksimum yang dapat diakses oleh *client* ke sumber tujuan.

2. *Delay (legacy)*

Delay merupakan waktu yang dibutuhkan untuk mengirim paket ke sumber tujuan (Patih, Fitriawan, dan Yuniati, 2012). Perhitungan nilai *delay* dimulai sejak awal paket terkirim hingga paket tersebut sampai ke sumber tujuan. *Delay* dapat dipengaruhi oleh jarak, media fisik, waktu proses yang lama. Kualitas layanan jaringan yang baik memiliki nilai *delay* yang kecil. *Delay* memiliki empat kategori latensi yaitu seperti Tabel 2.2 berikut.

Tabel 2.1 Kategori Latensi *Delay*

Kategori Latensi	<i>Delay</i>	Indeks
Sangat Bagus	< 150 ms	4
Bagus	150 ms s/d 300 ms	3
Sedang	300 ms s/d 450 ms	2
Jelek	> 450 ms	1

3. *Jitter*

Jitter merupakan perbedaan selang waktu kedatangan antar paket di terminal tujuan (Novianti & Widianoro, 2016). *Jitter* sering disebut variasi *delay*, jaringan yang stabil tidak memiliki *jitter* pada pengiriman pakatnya. *Jitter* dipengaruhi oleh variasi bebantrafik dan besar tumbukan antar paket dalam suatu jaringan. Semakin besar beban trafik dalam suatu jaringan akan menyebabkan semakin besar peluang terjadinya tumbukan sehingga nilai *jitter*nya juga

bertambah. *Jitter* memiliki empat kategori penurunan performansi jaringan yaitu seperti Tabel 2.3 berikut.

Tabel 2.2 Kategori Degradasi *Jitter*

Kategori Degradasi	<i>Jitter</i>	Indeks
Sangat Bagus	0 ms	4
Bagus	0 ms s/d 75 ms	3
Sedang	75 ms s/d 125 ms	2
Jelek	125 ms s/d 225 ms	1

4. *Packet Loss*

Packet loss didefinisikan sebagai kegagalan transmisi paket IP untuk mencapai tujuannya (Iskandar & Hidayat, 2015). kegagalan paket untuk mencapai tujuan dapat disebabkan oleh beberapa kemungkinan di antaranya *overload* trafik dalam jaringan, tabrakan (*congestion*), eror pada media fisik dan kegagalan pada sisi penerima. Dalam pengimplementasian jaringan, nilai nilai *packet loss* diharapkan mempunyai nilai yang minimum. Secara umum *packet loss* memiliki empat kategori penurunan performansi jaringan yaitu seperti pada Tabel 2.1 berikut.

Tabel 2.3 Kategori Degradasi *Packet Loss*

Kategori Degradasi	Persentase <i>Packet Loss</i>	Indeks
Sangat Bagus	0% - 2%	4
Bagus	3% - 14%	3
Sedang	15% - 24%	2
Jelek	> 25%	1

2.6.3 *Workload*

Workload merupakan total jumlah *request* yang dapat ditangani oleh *server* dalam satu waktu. Pengujian *workload* dilakukan untuk mengetahui kemampuan dari suatu *server* dalam menangani sejumlah *request* dari *client*. Hal ini bertujuan untuk mengetahui batas maksimal suatu *server* dalam menangani jumlah *request* dari *client* dalam memberikan pelayanan.

2.7 Media Pengujian

Dalam penelitian ini akan digunakan *tools* untuk membantu mendapatkan hasil analisa performansi terhadap pengujian yang akan dilakukan. Adapun *tools* pengujian yang dimaksud sebagai berikut.

2.8 Tools Pengujian

Tools yang digunakan untuk menguji *load server* adalah *siege*. *Siege* merupakan aplikasi yang dibuat untuk keperluan stress test dan benchmark pada *web server*. Stress test dan benchmarking diperlukan untuk mengetahui seberapa baik performa dari sebuah *web* dan seberapa banyak *request* yang bisa ditangani oleh *web* tersebut.

BAB III

METODOLOGI PENELITIAN

3.1 Analisis Kebutuhan Sistem

Analisis kebutuhan sistem bertujuan untuk mengdefenisikan kebutuhan agar sistem yang akan di uji berjalan dengan baik. Analisa yang akan dilakukan yaitu mendefenisikan kebutuhan perangkat keras dan perangkat lunak dari *server* dan laptop. Berdasarkan analisa kebutuhan yang telah dilakukan, berikut kebutuhan sistem yang didefenisikan dalam penelitian ini.

3.1.1 Kebutuhan Perangkat Keras Server

Perangkat keras *server* yang digunakan pada penelitian ini adalah perangkat keras yang dibutuhkan untuk mengimplementasikan *infrastructure as a service* pada *cloud computing*. Adapun kebutuhan perangkat keras *server* yang dapat terpenuhi, berikut deskripsi kebutuhan perangkat keras dari sisi *server*:

CPU	: 1x Intel® Xeon® E55606 4 Core 8 Threads @2.3 GHz
Memory	: 1x16 GB DDR3 PC 10600 ECC Reg
Harddisk	: 1x500 GB WD5000AAKX

3.1.2 Kebutuhan Perangkat Lunak Server

Server memerlukan beberapa perangkat lunak agar sistem dapat berjalan dengan baik. Perangkat lunak yang dibutuhkan terdiri dari sistem operasi, *database*, *web server*, *network*. Perangkat lunak tersebut dibutuhkan sebagai kondisi awal *host server* sebelum dilakukan konfigurasi *openstack* dan *load balancer*. Berikut kebutuhan perangkat lunak *server*:

Sistem Operasi	: CentOS 7
Database	: MariaDB
Web Server	: Httpd

3.1.3 Kebutuhan Perangkat Keras Laptop

Perangkat keras laptop difungsikan sebagai klien yang akan mengakses *instances* atau *virtual server*. Adapun spesifikasi dari laptop yang digunakan dalam penelitian ini sebagai berikut:

CPU : Intel® Core™ i5-7200U 2.5GHz
 Memory : 2x4 GB 2400 MHz
 Harddisk : SSD 250 GB M.2

3.1.4 Kebutuhan Perangkat Lunak Laptop

Perangkat lunak yang dibutuhkan dalam penelitian ini yaitu sistem operasi, *ssh client* dan *browser*. Berikut rincian kebutuhan perangkat lunak yang digunakan:

Sistem Operasi : Windows 10
 SSH Client : Putty dan MobaXterm
 Browser : Google Chrome

3.1.5 Kebutuhan Perangkat Keras Virtual Load Server

Perangkat keras yang dibutuhkan oleh *instance* menggunakan metode *load balancing* terdiri atas vCPU, *memory* dan *storage disk*. *Load server* adalah *instance* atau *virtual server* yang sudah diterapkan metode *load balancing*. Setiap *load server* yang dibuat akan diberikan konfigurasi perangkat keras yang sama. *Instance* yang memiliki *load balancer* dibuat dari gabungan dua *instance* menjadi satu. Berikut rincian kebutuhan perangkat keras yang digunakan pada *instance* yang memiliki *load balancer*:

vCPU : 1 Core
 Memory : 1024 MB
 Storage : 10 GB

3.1.6 Kebutuhan Perangkat Keras Virtual Database Server

Virtual database server adalah *server* yang digunakan untuk menyimpan data-data yang terkoneksi ke *load server* sebagai media penyimpanan dari *load server*. Berikut ini rincian dari kebutuhan perangkat keras *virtual database server*:

vCPU	: 1 Core
Memory	: 2048 MB
Storage	: 20 GB

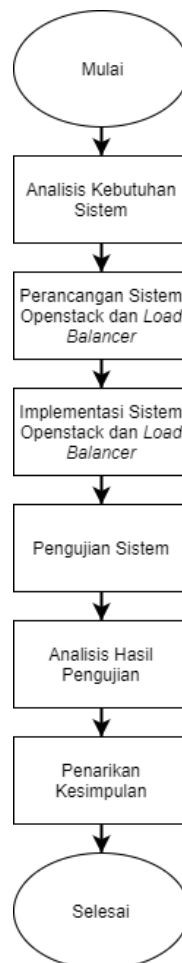
3.1.7 Kebutuhan Perangkat Lunak *Virtual Server*

Virtual server yang terdapat pada openstack menggunakan *template* dari *default template* openstack. Perangkat lunak yang dibutuhkan terdiri dari *web server* dan *database*. Berikut ini rincian dari kebutuhan perangkat lunak *instance*:

Sistem Operasi	: Ubuntu Server 16.04 Xenial Xerus
Web server	: Apache2
Database	: mysql-server/mysql-client

3.2 Metodologi Penelitian

Metodologi penelitian yang dilakukan dapat dijelaskan pada diagram alir penelitian pada gambar 3.1



Gambar 3.1 Diagram alir penelitian

1. Analisis Kebutuhan Sistem

Pada tahap awal yang akan dilakukan pada penelitian ini adalah mengidentifikasi kebutuhan sistem yang akan dibangun agar dapat berjalan dengan semestinya. Identifikasi yang dilakukan baik terhadap perangkat keras atau perangkat lunak. Analisis kebutuhan merupakan tahap dimana kebutuhan-kebutuhan sistem yang akan dibuat dianalisa terlebih dahulu sebelum masuk ke tahap perancangan sistem. Pada tahap ini akan ditentukan beutuhan apa saja yang diperlukan dalam penelitian agar peneliti dapat berjalan sebagaimana semestinya.

2. Perancangan Sistem Openstack dan *Load Balancer*

Perancangan sistem openstack dan *load balancer* dimulai dengan mengidentifikasi kebutuhan sistem dan analisa kebutuhan. Dari hasil identifikasi tersebut akan dilakukan pemetaan jaringan terdahap sistem yang akan dibuat. Kemudian akan dilakukan perancangan yang diperlukan untuk membangun openstack dan *load balancer*. Perancangan sistem openstck dan *load balancer* terdapat perancangan arsitektur jaringan.

3. Implementasi Sistem Openstack dan *Load Balancer*

Pada tahap ini, sistem akan diimplementasikan berdasarkan perancangan sistem openstack dan *load balancer* yang telah dibuat. Implementasi dimulai dengan konfigurasi jaringan, kemudian konfigurasi openstack dan *load balancer* pada *server*.

4. Pengujian Sistem

Pengujian sistem dilakukan dengan menggunakan metode *availability*, *quality of service*, dan *workload* untuk menguji kemampuan *instance* menggunakan metode *load balancing* pada kondisi jaringan lokal, jaringan sepi dan jaringan sibuk. Pada tahap ini juga dilakukan pengambilan data untuk setiap pengujian yang dilakukan.

5. Analisis Hasil Pengujian

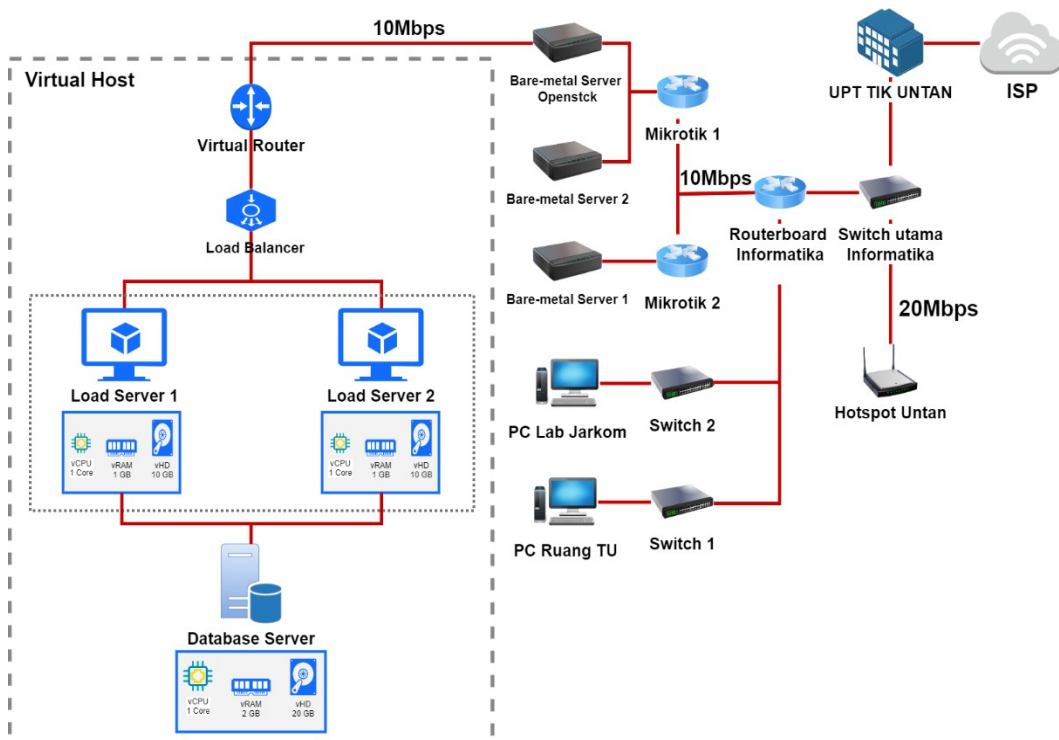
Analisis hasil pengujian dilakukan setelah proses pengujian yang dilakukan pada *load server*. Pada tahapan ini, data yang diambil dari berbagai aspek pengujian yang telah dilakukan akan ditampilkan dalam bentuk tabel dan grafik. Analisa dilakukan dengan melihat perkembangan grafik pengujian *load server* pada aspek yang dilakukan pengujian.

6. Penarikan Kesimpulan

Penarikan kesimpulan merupakan tahap terakhir setelah dilakukan analisa hasil pengujian. Pada tahap ini, kesimpulan akan dibuat berdasarkan hasil dari analisis pengujian. Perbandingan performa *load server* akan dibahas berdasarkan grafik yang telah dibuat.

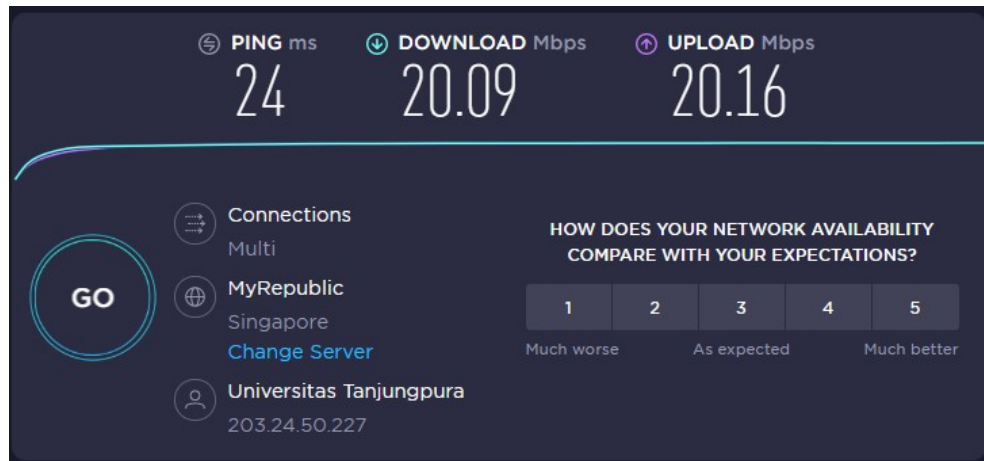
3.3 Perancangan Sistem Openstack dan Load Balancer

Perancangan sistem openstack dan *load balancer* dimulai dari perancangan arsitektur jaringan untuk menjelaskan bentuk dari implementasi jaringan yang akan dilakukan. Perancangan aksitektur jaringan akan diterapkan pada jaringan informatika dijelaskan pada gambar 3.2.



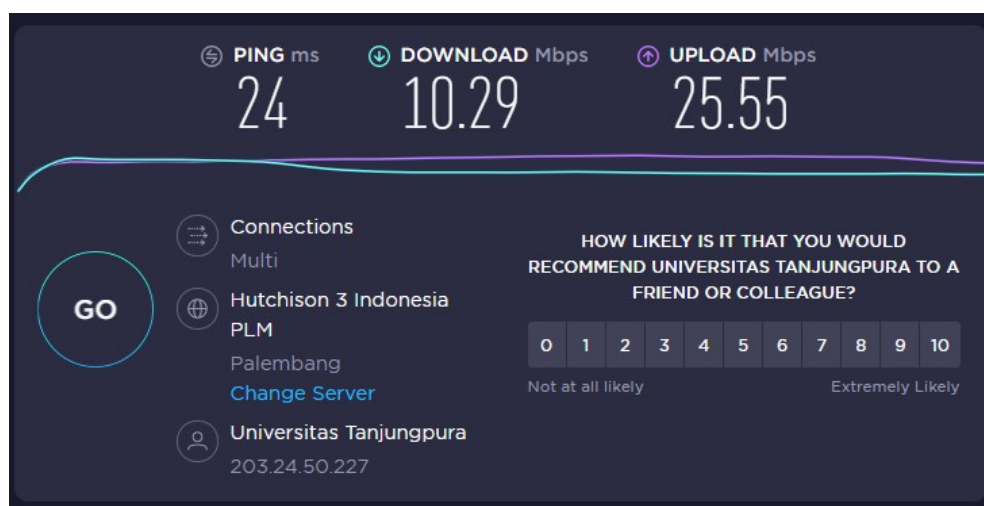
Gambar 3.2 Perancangan arsitektur jaringan

Gedung informatika mendapatkan akses internet dari UPT TIK Untan sebesar 20 Mbps untuk jaringan *wireless*, kemudian untuk *server* mendapat akses internet sebesar 10 Mbps. Akses internet sebesar 20 Mbps yang diperoleh kemudian diteruskan ke seluruh ruangan Gedung informatika melalui satu *accesspoint* yang terdapat pada Gedung informatika terdapat pada gambar 3.3.



Gambar 3.3 Bandwith jaringan wireless

Akses internet sebesar 10 Mbps yang diperoleh dari *routerboard* kemudian dihubungkan ke dua mikrotik yang ada pada Gedung informatika, yang mana masing-masing mikrotik mendapatkan akses internet sebesar 10 Mbps. Pengembangan jaringan untuk penelitian menggunakan salah satu mikrotik agar bisa diakses dari jaringan lokal dan jaringan public terdapat pada gambar 3.4.

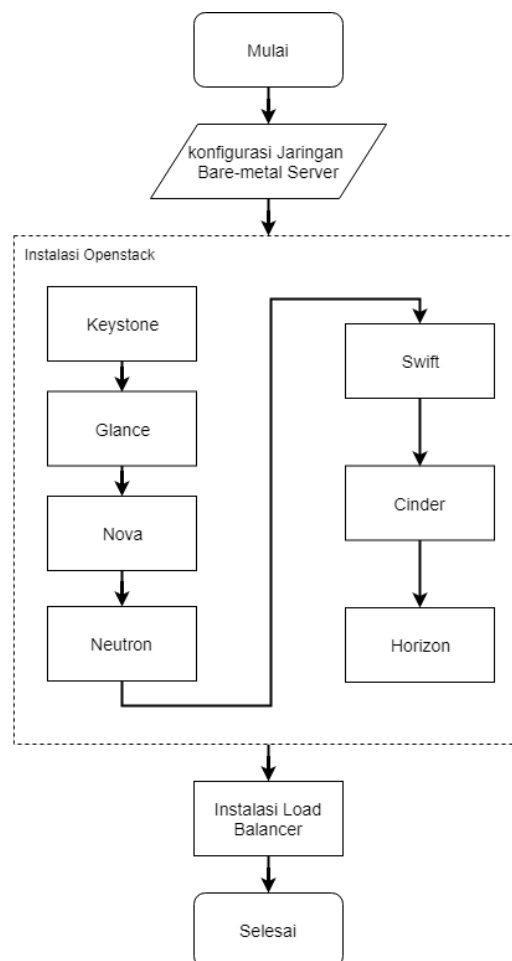


Gambar 3.4 Bandwith jaringan lokal

Kemudian *bare-metal server* yang digunakan untuk mengimplementasikan openstack dilakukan perancangan yang akan membuat suatu buah *virtual router*, dua buah *instance* untuk *load server* dan tiga jenis konfigurasi *load balancer* yang didalamnya terdapat algoritma *load balancer* yaitu *round robin*, *least connection* dan *source ip*.

3.4 Implementasi Sistem Openstack dan Load Balancer

Implementasi sistem openstack dan *load balancer* memerlukan beberapa *service* dasar diantaranya *network*, *identity service*, *image service*, *compute* dan *load service*. Selain *service* dasar, diperlukan juga beberapa *package* lain untuk mendukung openstack agar bekerja lebih baik. Dalam implementasi sistem openstack berbasis *load balancer*, agar instalasi berjalan dengan baik maka proses instalasi dimulai dari konfigurasi *bare-metal server*, instalasi openstack dan instalasi *load balancer* yang dilakukan sesuai dengan diagram alir pada gambar 3.5.



Gambar 3.5 Diagram alir instalasi openstack dan *load balancer*

Untuk menjalankan stack dasar pada openstack, diperlukan perisapan *server* dan beberapa *service* yang digunakan dalam perancangan sistem sebagai berikut:

1. Konfigurasi jaringan *bare-metal server* merupakan konfigurasi awal sebelum dilakukan instalasi openstack. Konfigurasi jaringan *bare-metal server* terdiri dari konfigurasi *ip address* yang disetting secara *static*.
2. Keystone adalah komponen yang menyediakan outentikasi di semua infrastruktur *cloud*, jadi semua komponen dari *core* dan *optional* akan melewati keystone untuk diverifikasi.
3. Glance berfungsi untuk memanajemen *disk image*. *Disk image* ini nantinya diberikan ke *instance* pada *virtual server* lewat nova, dan digunakan untuk membuat hardisk virtual.
4. Nova merupakan komponen untuk mengatur proses alokasi CPU untuk setiap *virtual machine*, dan nova merupakan komponen yang utama dalam setiap sistem *infrastructure as a service*.
5. Neutron merupakan komponen yang menyediakan *network connectivity as a service* yang mengatur jaringan di openstack, seperti *subnet*, *routing*, *load-balancer*, *gateway*, dan *floating IP*.
6. Swift fungsinya sama seperti cinder, bedanya di swift dapat menyimpan data tak terhingga sedangkan cinder lebih seperti *harddisk* di laptop atau pc, yang bersifat virtual.
7. Cinder merupakan komponen yang berfungsi menyediakan *block storage* yang akan dipakai oleh komponen *instances*.
8. Horizon merupakan tampilan *dashboard* yang menyediakan *web interface* untuk mengontrol semua komponen yang ada didalam openstack.
9. *Load balancer* merupakan suatu fitur yang bisa ditambahkan ke dalam openstack, *load balancer* berfungsi untuk mendistribusikan banyak *request* pada dua atau lebih *instance* secara seimbang, agar *instance* dapat berjalan optimal.

Setelah proses instalasi selesai maka akan dilanjutkan dengan proses konfigurasi jaringan, konfigurasi jaringan *internal*, konfigurasi *load balancer* dan konfigurasi *instance load server*.

3.4.1 Konfigurasi Jaringan Openstack

Pada bagian ini akan dibahas mengenai implementasi jaringan openstack yang ada pada sistem openstack. Tujuan implementasi jaringan openstack yaitu agar *instance* yang akan dibuat bisa terhubung ke jaringan luar dan bisa mendapat akses internet sehingga *instance* yang dibuat dapat diakses melalui jaringan publik. Konfigurasi jaringan pada openstack terdiri dari beberapa tahapan, seperti membuat jaringan *internal* dan *external*, membuat *virtual router*, dan konfigurasi *security group*.

1. Membuat Jaringan Openstack

Konfigurasi jaringan openstack yang akan dibuat memiliki spesifikasi sesuai dengan tabel berikut.

Tabel 3.1 Spesifikasi Jaringan Openstack

Jaringan	Subnet	Tipe
Public_Network	192.168.100.0/24	Public
Private_Network	10.0.1.0/24	Private

Jaringan yang terdapat pada openstack memiliki dua jenis jaringan yaitu jaringan *public* dan jaringan *private*. Jaringan *public* pada openstack adalah jaringan yang menghubungkan antara jaringan diluar sistem openstack dengan jaringan *private* sehingga *instance* yang dibuat bisa mendapatkan akses internet dan bisa diakses lewat ip yang terdapat pada mikrotik. Jaringan *public* pada openstack akan mendapatkan ip sesuai dengan konfigurasi port pada mikrotik yang biasa disebut *floating ips*. Jaringan *private* pada openstack berfungsi untuk memberikan ip lokal pada *instance* yang akan dibuat. Langkah-langkah untuk membuat jaringan *public* dan *private* pada openstack dapat dilihat pada gambar 3.6.

The image displays two screenshots of the OpenStack 'Create Network' form. The left screenshot shows the 'Network' tab with the following fields and options:

- Name:** Public_Network
- Project:** admin (with a red border and the message 'This field is required.')
- Provider Network Type:** Flat
- Physical Network:** extnet
- Options:**
 - ☒ Enable Admin State
 - ☐ Shared
 - ☒ External Network
 - ☒ Create Subnet
- Availability Zone Hints:** nova

The right screenshot shows the 'Subnet' tab with the following fields and options:

- Subnet Name:** PubNet
- Network Address:** 192.168.100.0/24
- IP Version:** IPv4
- Gateway IP:** 192.168.100.1
- Options:**
 - ☐ Disable Gateway
 - ☒ Enable DHCP
 - Allocation Pools:** 192.168.100.0, 192.168.100.200
 - DNS Name Servers:** (empty)
 - Host Routes:** (empty)

Gambar 3.6 Membuat jaringan *public* dan *private*

2. Membuat *Virtual Router*

Virtual router pada openstack berfungsi untuk menghubungkan ip lokal pada tiap *instance* dengan ip public agar setiap *instance* dapat mengakses dan diakses oleh *client*. Langkah untuk membuat *virtual router* terdapat pada gambar 3.7.

Gambar 3.7 Membuat *virtual router*

3. Konfigurasi *Security Group*

Security group adalah aturan yang digunakan dalam sistem openstack untuk membatasi lalu lintas jaringan yang memiliki akses ke *instance*. *Security group* yang dibuat sesuai dengan tabel berikut.

Tabel 3.2 Konfigurasi *Security Group*

Arah	Jenis Eter	Protocol IP	Rentang Port
Keluar	IPv4	ICMP	Bebas
Keluar	IPv4	TCP	1 - 65535
Keluar	IPv4	UDP	1 - 65535
Masuk	IPv4	ICMP	Bebas
Masuk	IPv4	TCP	1 - 65535
Masuk	IPv4	UDP	1 - 65535

Security group ini berupa aturan-aturan yang dibuat untuk membatasi akses *client* sesuai dengan hak akses yang dibuat. Langkah untuk membuat *security group* terdapat pada gambar 3.8.

Add Rule

Rule ^{*}

Custom TCP Rule ▼

Description [?]

Direction

Ingress ▼

Open Port ^{*}

Port ▼

Port ^{*} [?]

Remote ^{*} [?]

CIDR ▼

CIDR [?]

0.0.0.0/0

Gambar 3.8 Membuat *security group*

3.4.2 Konfigurasi *Load Balancer*

Load balancer adalah salah satu metode yang digunakan untuk mendistribusikan jumlah *request* yang masuk secara bersamaan terhadap *web server* untuk meminimalisir terjadinya kegagalan *request* dari *client* agar *server* bekerja lebih optimal. Pada sistem openstack sudah disediakan tiga metode *load balancer* yang dapat digunakan seperti: *round robin*, *least connection* dan *source ip*. Berikut adalah langkah-langkah untuk membuat *load balancer* yang terdapat pada gambar 3.9.

Create Load Balancer

Load Balancer Details ^{*}

Listener Details ^{*}

Pool Details ^{*}

Pool Members

Monitor Details ^{*}

Provide the details for the load balancer.

Name

Load Balancer 4

Description

IP address

Subnet ^{*}

< Back

Next >

Create Load Balancer

Gambar 3.9 Membuat *load balancer*

3.4.3 Konfigurasi *Instance Load Server*

Instance merupakan mesin virtual yang berjalan pada sistem *cloud*. Untuk menjalankan *instance* sendiri diperlukan beberapa komponen utama seperti *image*, *flavors* dan *network*. *Image* merupakan jenis sistem operasi yang akan digunakan ketika membuat sebuah *instance*. *Flavors* berfungsi untuk menentukan jumlah sumberdaya seperti kapasitas vCPU, RAM dan *harddisk* yang akan digunakan oleh *instance*. *Network* merupakan jaringan yang akan digunakan ketika ingin membuat sebuah *instance*, *ip address* pada tiap *instance* ditentukan secara dinamis oleh sistem. Berikut adalah langkah untuk membuat sebuah *instance* terdapat pada gambar 3.10.

Gambar 3.10 Konfigurasi *instance load server*

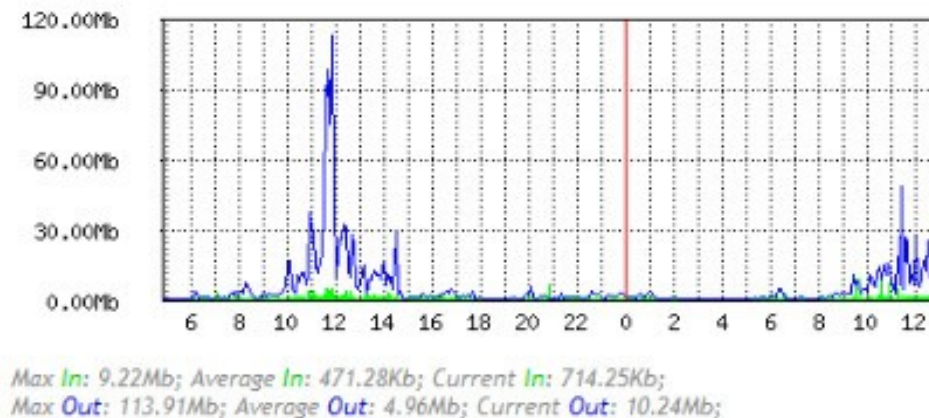
3.5 Pengujian Sistem

Pada tahap ini dilakukan pengujian terhadap *web server* apakah sistem *load balancer* pada *openstack* dapat berfungsi sesuai dengan perancangan atau tidak. Tahapan pengujian dilakukan untuk memastikan bahwa sistem yang telah dibuat dapat berjalan sesuai dengan perancangan pengujian pada *load server*. Pengujian akan dilakukan pada tiga kondisi jaringan yaitu jaringan lokal, jaringan sepi dan jaringan sibuk yang terdapat pada gambar 3.11 sesuai dengan jaringan yang terdapat di untan.

Interface <FT-IF-144> Statistics

• Last update: Tue Jan 21 12:48:11 2020

"Daily" Graph (5 Minute Average)



Gambar 3.11 Grafik penggunaan jaringan untan

Jaringan lokal adalah jaringan yang dilakukan pengujian terhadap *web server* yang terdapat pada satu kelas ip dengan *virtual server*. Sedangkan jaringan sepi adalah pengujian yang dilakukan terhadap *web server* pada jaringan untan2018 pada jam 02:00 hingga jam 04:00 dini hari ketika jaringan pada trafik yang rendah dan stabil. Pengujian pada jaringan sibuk adalah pengujian yang dilakukan pada jaringan untan2018 dengan kondisi trafik tinggi dan tidak stabil pada jam 10:00 hingga jam 12:00 siang hari. Adapun pengujiannya yakni sebagai berikut.

3.5.1 Pengujian Availability

Pengujian *availability* adalah pengujian terhadap ketersediaan layanan dari *web server*. Pada pengujian ini akan dilihat apakah *web server* dapat memberikan layanan ketika terjadi *down*. Pengujian *availability* bertujuan untuk melihat seberapa optimal kinerja layanan yang dapat diberikan oleh *web server* meskipun terjadi kegagalan pada *web server* melalui beberapa skenario pengujian. Adapun skenario pengujian yang akan dilakukan sebagai berikut.

1. Skenario 1: pengujian dilakukan pada kondisi kedua *web server* dalam kondisi menyala.
2. Skenario 2: pengujian dilakukan pada kondisi kedua *web server* dalam keadaan menyala kemudian salah satu *web server* dimatikan.
3. Skenario 3: pengujian dilakukan pada kondisi kedua *web server* dalam keadaan tidak menyala, dan salah satu dihidupkan.

Setelah mendapatkan hasil pengujian *downtime*, selanjutnya akan dilakukan perhitungan nilai *uptime* dari setiap pengujian akan mendapatkan hasil *availabnility* melalui persamaan berikut.

$$Uptime = Lama Waktu setiap kali Pengujian - Downtime \quad (3.2)$$

$$Availability = \frac{uptime}{uptime + downtime} \times 100\% \quad (3.3)$$

Hasil perhitungan *availability* melalui persamaan diatas akan ditampilkan pada tabel 3.4 berikut.

Tabel 3.4 Hasil Perhitungan *Availability*

Server	Skenari o	Availability (%)										Rata- rata
		ke-1	ke-2	ke-3	ke-4	ke-5	ke-6	ke-7	ke-8	ke-9	ke-10	
Round Robin	1											
	2											
	3											
Least Connections	1											
	2											
	3											
Source IP	1											
	2											
	3											

Pengujian *availability* dilakukan dengan menggunakan *tools* siege dengan perintah untuk melakukan *benchmark* ke *web server* yang akan dilakukan pengujian. Hal ini dimaksudkan untuk mendapatkan nilai *downtime* dan *uptime*. Adapun perintah yang dijalankan sebagai berikut.

```
Siege -b -c [request] -r [perulangan] [ip/domain]
```

Berdasarkan perintah diatas maka akan dilakukan pengujian *web server* dengan domain atau ip dari *web*. Kemudian akan dilakukan *benchmark* dengan *command -b*, banyaknya *request* dikirim dalam satu kali pengujian dengan *command -c, -r* menandakan banyaknya perulangan dari pengiriman *request*.

3.5.2 Pengujian *Quality of Service*

Pengujian *quality of service* dilakukan untuk melihat kemampuan dari sebuah jaringan dalam menyediakan layanan yang baik bagi layanan trafik yang melewatinya. Untuk pengujian *quality of service* menggunakan *tools siege* yang berfungsi untuk memonitoring dan mengambil data pengujian. Dari data pengujian tersebut akan dilakukan perhitungan untuk mengetahui kemampuan dari sebuah jaringan dalam menyediakan layanan. Pengujian terhadap kemampuan dari sebuah layanan jaringan menggunakan beberapa parameter yang terdapat pada *quality of service*. Parameter yang terdapat pada *quality of service* yaitu *throughput, packet loss, delay* dan *jitter*.

1. Pengujian Throughput

Pengujian *throughput* dilakukan dengan melakukan pengukuran *throughput* pada *load server*. *Throughput* digunakan untuk mengetahui kemampuan *web server* dalam memberikan layanan secara benar terhadap *request* yang datang secara bersamaan. Data hasil pengamatan selanjutnya akan dilakukan perhitungan terhadap *throughput*. Nilai *throughput* diperoleh dengan menggunakan persamaan dan ditampilkan dalam tabel berikut.

$$\text{Throughput} = \frac{\text{paket data yang diterima}}{\text{lama pengamatan}} \quad (3.4)$$

Tabel 3.5 Hasil Perhitungan *Throughput*

<i>Request</i>	<i>Throughput (MBps)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100			
200			
300			

<i>Request</i>	<i>Throughput (MBps)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
...			
1500			
Rata-rata			

2. Pengujian *Delay*

Pengujian *delay* dilakukan dengan mengamati jumlah paket yang diterima selama pengujian. *Web server* akan diberikan beban akses dengan jumlah yang ditentukan secara bervariasi yaitu dengan mengirimkan beban akses kepada *web server*. Pengujian *delay* menggunakan *tools siege* yang akan *mengcapture* data pengamatan selama pengujian. Hasil pengujian *delay* akan dimuat dalam tabel dan menggunakan persamaan berikut.

$$\text{Delay rata-rata} = \frac{\text{lama pengamatan}}{\text{paket diterima}} \quad (3.5)$$

Tabel 3.6 Hasil Perhitungan *Delay*

<i>Request</i>	<i>Delay (ms)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100			
200			
300			
...			
1500			
Rata-rata			

3. Pengujian *Jitter*

Pengujian *jitter* dilakukan dengan melakukan pengamatan pada jumlah paket yang diterima selama pengujian. Nilai *delay* juga diperhatikan selama pengujian yang akan digunakan untuk mendapatkan nilai *jitter*. Selama pengujian *web server* akan diberikan beban akses dengan jumlah yang ditentukan secara bervariasi. Pengujian *jitter* menggunakan *tools siege* yang akan *mengcapture* data pengamatan selama pengujian. Hasil pengujian *jitter* akan dimuat dalam tabel dan akan dilakukan perhitungan terhadap nilai *jitter* menggunakan persamaan berikut.

$$\text{Total variasi delay} = \text{lama pengamatan} - \text{delay} \quad (3.6)$$

$$\text{Jitter} = \frac{\text{total variasi delay}}{\text{total paket diterima}} \quad (3.7)$$

Tabel 3.7 Hasil Perhitungan *Jitter*

<i>Request</i>	<i>Jitter (ms)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100			
200			
300			
...			
1500			
Rata-rata			

4. Pengujian Packet loss

Pengujian *packet loss* dilakukan dengan pengamatan pada jumlah paket yang hilang disaat terjadinya komunikasi antara *web server* dan *client*. Pengujian *packet loss* dilakukan untuk melihat seberapa besar jumlah paket yang hilang saat *web server* menangani permintaan dari *client*. Pengujian dilakukan dengan membandingkan *web server* pada *load server*. *Web server* akan diberikan beban akses dengan jumlah yang ditentukan secara berkala. Pengujian *packet loss* menggunakan *tools siege*. Selama pengujian *tools siege* akan *mengcapture* jumlah paket yang masuk sehingga diperoleh data hasil pengujian yang akan dilakukan perhitungan dengan menggunakan persamaan dan ditampilkan dalam tabel berikut.

$$\text{Packet Loss} = \frac{(\text{paket dikirim} - \text{paket diterima})}{\text{paket dikirim}} \times 100\% \quad (3.8)$$

Tabel 3.8 Hasil Perhitungan *Packet Loss*

<i>Request</i>	<i>Packet Loss (%)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100			
200			
300			
...			
1500			
Rata-rata			

3.5.3 Pengujian *Workload*

Pengujian *workload* dilakukan menggunakan *tool* siege dengan melakukan *benchmark* terhadap *web server*. Pengujian *workload* dilakukan dengan cara memberikan *request* dengan jumlah yang ditentukan hingga *server* tidak dapat menangani semua *request tersebut*. Pemberian *request* pada pengujian *workload* hampir sama dengan pengujian *quality of service* hanya saja dibatasi dengan jumlah maksimal *request* yang diberikan terhadap *server*, sehingga *server* masih bisa menangani *request*. Pengujian ini dilakukan pada *load server* sehingga akan terlihat perbedaan kemampuan metode dari *load balancer* dalam menangani *request* secara maksimal. Hasil pengujian *workload* akan dirangkum dalam tabel berikut.

Tabel 3. 9 Hasil Pengujian *Workload*

Jumlah <i>Request</i>	<i>Workload</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100			
200			
300			
...			

3.6 Analisis Hasil Pengujian

Pada tahap ini akan dilakukan analisis hasil pengujian yang telah dilakukan pada tahap sebelumnya. Dari hasil pengujian akan dilakukan perbandingan antar *load server* yang menggunakan metode *round robin*, *least connection*, *source ip* sehingga akan terlihat perbedaan dari *server* tersebut.

3.7 Penarikan Kesimpulan

Berdasarkan hasil analisis pengujian yang telah dilakukan maka akan ditarik kesimpulan mengenai apakah sistem yang telah dibangun dapat berjalan sesuai dengan apa yang diharapkan. Penarikan kesimpulan dilakukan berdasarkan kepada tujuan dilakukan penelitian sehingga hasil dari penarikan kesimpulan merupakan jawaban dari pertanyaan yang disampaikan pada tujuan dilakukan penelitian.

BAB IV

IMPLEMENTASI DAN ANALISIS

4.1 Implementasi

Implementasi dilakukan berdasarkan perancangan yang telah dibuat. Implementasi sistem menggunakan *bare-metal server* sebagai *controller* dan *compute node* pada openstack. Manajemen *instance*, *flavors*, *image*, *volume*, *security group*, *networks* dan *load balancers* dilakukan melalui *Graphical user Interface* (GUI) berbasis *web*.

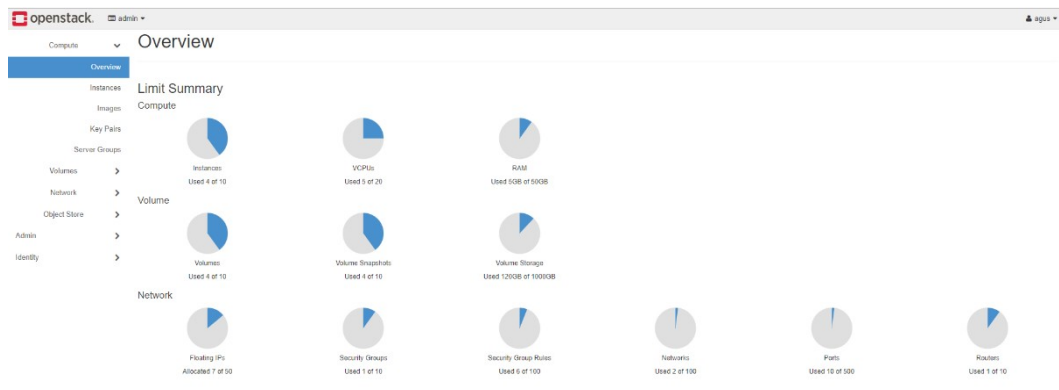
4.2 Antarmuka Openstack

4.2.1 Antarmuka Compute

Antarmuka *compute* openstack merupakan antarmuka yang digunakan untuk mengelola berbagai layanan yang terdapat pada openstack yang berkaitan langsung dengan *server virtual*. Antarmuka *compute* terdiri dari beberapa antarmuka seperti, *overview*, *instances*, *images*, *key pair* dan *flavors*.

4.2.1.1 Antarmuka Overview

Antarmuka *overview* openstack menampilkan jumlah keseluruhan dari penggunaan *compute*, *volume* dan *network*. *Compute* terdiri dari *instances*, VCPUs, RAM. *Volume* terdiri dari *volumes*, *volume snapshots*, *volume storage* sedangkan *network* terdiri dari *floating ips*, *security group*, *security group rules*, *networks*, *ports* dan *routers*. Untuk penelitian ini, *compute* keseluruhan yang digunakan terdiri dari 4 *instances*, 5 VCPUs, 5 GB RAM. Jumlah *volume* keseluruhan yang digunakan sebanyak 4 *volumes*, 4 *volume snapshots* dan 120 GB *volume storage*. Jumlah *network* keseluruhan yang digunakan sebanyak 7 *floating ips*, 1 *security group*, 6 *security group rules*, 2 *networks*, 10 *ports* dan 1 *routers*. Untuk lebih jelasnya dapat dilihat pada Gambar 4.1 Antarmuka *overview* openstack.



Gambar 4.1 Antarmuka overview openstack

4.2.1.2 Antarmuka Instance

Antarmuka *instances* openstack dapat menampilkan dan mengelola layanan *instance*. pada antarmuka *instances*, pengguna dapat melakukan berbagai kativitas seperti, menambah, menghapus, menghidupkan dan mematikan *instance*. Dalam penelitian ini dibuat tiga *instance* yang terdiri dari satu *database server*, dan dua *load server*. Dalam pengujian penelitian ini akan dilakukan pengujian pada *web server*. Tampilan antarmuka *instances* dapat dilihat pada Gambar 4.2 Antarmuka *instances*.

openstack

admin

agus

Project

API Access

Compute

Overview

Instances

Images

Key Pairs

Server Groups

Volumes

Network

Object Store

Admin

Identity

Project / Compute / Instances

Instances

Instance ID

Filter

Launch Instance

Delete Instances

More Actions

Displaying 4 items

<input type="checkbox"/>	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions	
<input type="checkbox"/>	Database	Database Utama	10.0.1.20 Floating IPs: 192.168.100.12	Database	single-keypair	Active	us-east-1a	nova	None	Running	0 minutes	Create Snapshot
<input type="checkbox"/>	Load 2	Load 2	10.0.1.21 Floating IPs: 192.168.100.18	Load Server	single-keypair	Active	us-east-1a	nova	None	Running	0 minutes	Create Snapshot
<input type="checkbox"/>	Single Server	Single Server	10.0.1.13 Floating IPs: 192.168.100.16	Single Server	single-keypair	Active	us-east-1a	nova	None	Running	0 minutes	Create Snapshot
<input type="checkbox"/>	Load 1	Load 1	10.0.1.23 Floating IPs: 192.168.100.17	Load Server	single-keypair	Active	us-east-1a	nova	None	Running	0 minutes	Create Snapshot

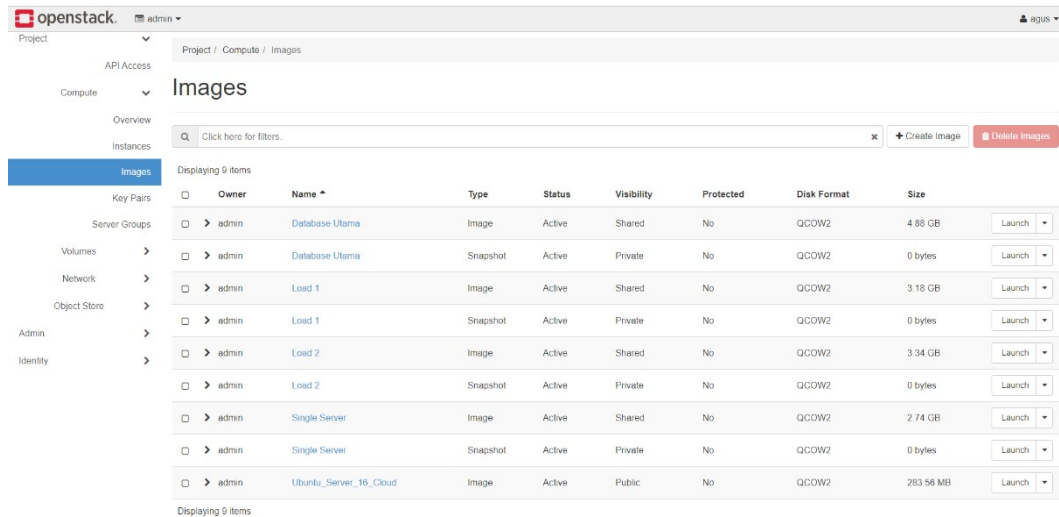
Displaying 4 items

Gambar 4.2 Antarmuka *instances*

4.2.1.3 Antarmuka Images

Antarmuka *images* openstack berfungsi untuk menampilkan dan mengelola *images*. Pengguna dapat menambahkan, mengubah dan menghapus

images. *Images* pada openstack berfungsi sebagai sistem operasi yang akan digunakan pada saat membuat *instances*. Pada penelitian ini *images* yang dibuat menggunakan ubuntu *server* berbasis *cloud* serta *image database*, *load server* dibuat dari hasil snapshot dari tiap *instance* yang dibuat pada penelitian. Berikut tampilan antarmuka *images* dapat dilihat pada Gambar 4.3 Antarmuka *images*.



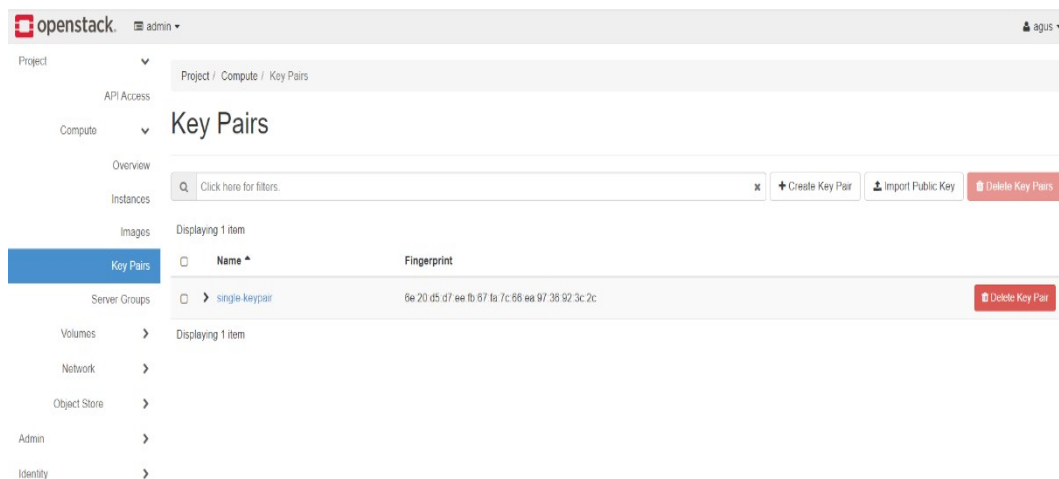
The screenshot shows the OpenStack Images dashboard. The left sidebar contains navigation links: Project, API Access, Compute, Overview, Instances, Images (selected), Key Pairs, Server Groups, Volumes, Network, Object Store, Admin, and Identity. The main content area is titled 'Images' and displays a table of 9 items. The table has columns: Owner, Name, Type, Status, Visibility, Protected, Disk Format, and Size. Each row has a 'Launch' button.

Owner	Name	Type	Status	Visibility	Protected	Disk Format	Size
admin	Database Utama	Image	Active	Shared	No	QCOW2	4.88 GB
admin	Database Utama	Snapshot	Active	Private	No	QCOW2	0 bytes
admin	Load 1	Image	Active	Shared	No	QCOW2	3.18 GB
admin	Load 1	Snapshot	Active	Private	No	QCOW2	0 bytes
admin	Load 2	Image	Active	Shared	No	QCOW2	3.34 GB
admin	Load 2	Snapshot	Active	Private	No	QCOW2	0 bytes
admin	Single Server	Image	Active	Shared	No	QCOW2	2.74 GB
admin	Single Server	Snapshot	Active	Private	No	QCOW2	0 bytes
admin	Ubuntu_Server_16_Cloud	Image	Active	Public	No	QCOW2	263.56 MB

Gambar 4.3 Antarmuka *images*

4.2.1.4 Antarmuka *Key Pairs*

Antarmuka *key pair* openstack dapat menampilkan dan mengelola *key pair*. *Key pair* berfungsi sebagai hak akses seperti *username* dan *password* yang sudah terenkripsi yang digunakan untuk *remote instance* yang beroperasi. Dalam penelitian ini menggunakan satu *key pair* yang digunakan pada semua *instances*. Tampilan antarmuka *key pairs* dapat dilihat pada Gambar 4.4 Antarmuka *key pairs*.



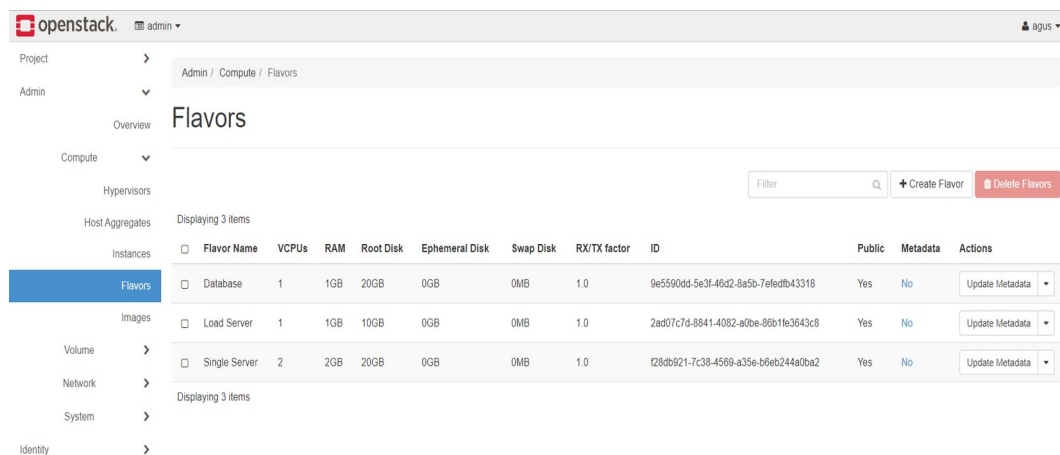
The screenshot shows the OpenStack Key Pairs dashboard. The left sidebar contains navigation links: Project, API Access, Compute, Overview, Instances, Images, Key Pairs (selected), Server Groups, Volumes, Network, Object Store, Admin, and Identity. The main content area is titled 'Key Pairs' and displays a table with 1 item. The table has columns: Name and Fingerprint. Each row has a 'Delete Key Pair' button.

Name	Fingerprint
single-keypair	6e 20 d5 d7 ee fb 67 1a 7c 66 ee 97 38 92 3c 2c

Gambar 4.4 Antarmuka *key pairs*

4.2.1.5 Antarmuka *Flavors*

Antarmuka *flavors* openstack menampilkan dan mengelola *flavor*. *Flavors* merupakan layanan pada openstack yang berfungsi untuk mengatur kapasitas vCPUs, RAM, dan *disk* yang akan digunakan *instances*. Pada penelitian ini terdapat tiga *flavor* yang dibuat seperti, *flavor database* dan *flavor load server*. Antarmuka *flavors* dapat dilihat pada Gambar 4.5 Antarmuka *flavors*.



Flavor Name	VCPUs	RAM	Root Disk	Ephemeral Disk	Swap Disk	RX/TX factor	ID	Public	Metadata	Actions
Database	1	1GB	20GB	0GB	0MB	1.0	9e5590dd-5e3f-46d2-8a5b-7efedfb43318	Yes	No	Update Metadata
Load Server	1	1GB	10GB	0GB	0MB	1.0	2ad07c7d-8841-4082-a0be-86b1fe3643c8	Yes	No	Update Metadata
Single Server	2	2GB	20GB	0GB	0MB	1.0	f28db921-7c38-4569-a35e-b6eb244a0ba2	Yes	No	Update Metadata

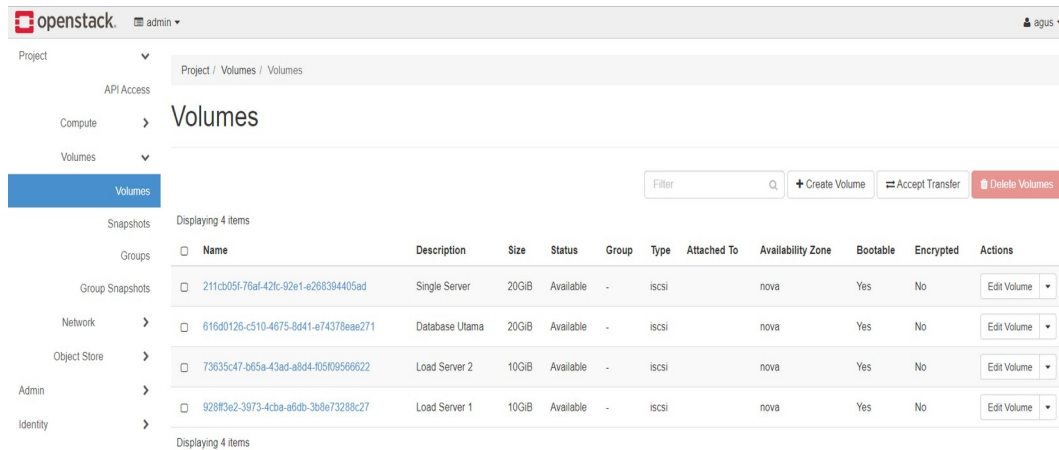
Gambar 4.5 Antarmuka *flavors*

4.2.2 Antarmuka *Volume*

Antarmuka *volumes* openstack merupakan antarmuka yang digunakan untuk mengelola berbagai layanan yang terdapat pada openstack yang berkaitan langsung dengan penyimpanan. Antarmuka *volumes* terdiri dari beberapa antarmuka seperti *volumes* dan *snapshots*.

4.2.2.1 Antarmuka *Volumes*

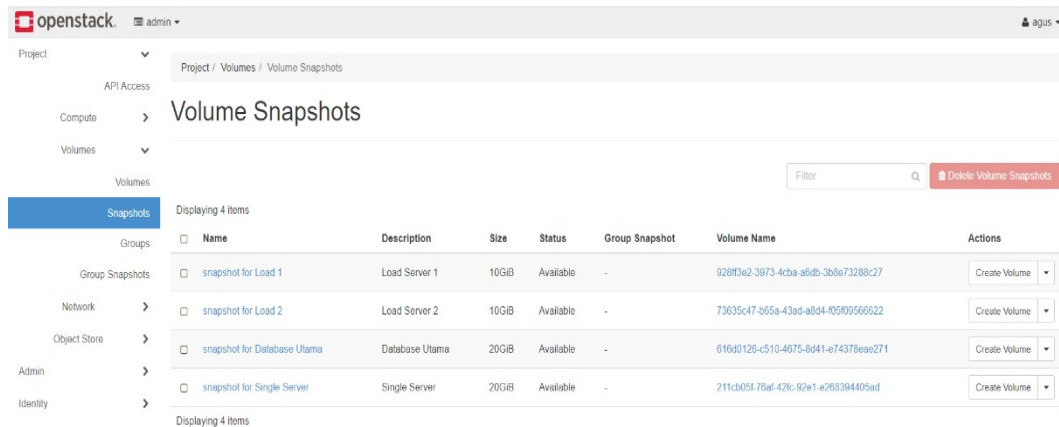
Antarmuka *volumes* openstack menampilkan dan mengelola *volumes*. *Volumes* merupakan penyimpanan yang digunakan oleh *instances*. Pengguna dapat membuat *volume* baru dan menambahkan ke *instances* yang telah dibuat. *Volume* pada openstack dapat dijadikan *images* dan membuat *instance* baru dari *volume* yang digunakan oleh *instance* lain, hal ini memudahkan pengguna untuk memperbanyak jumlah *instance* dengan *volume* yang sudah dikonfigurasi, tanpa harus mengkonfigurasi ulang. Tampilan antarmuka *volumes* dapat dilihat pada Gambar 4.6 Antarmuka *volumes*.



Gambar 4.6 Antarmuka volumes

4.2.2.2 Antarmuka Snapshots

Antarmuka *snapshot* openstack menampilkan dan mengelola *snapshots*. *Snapshots* merupakan Salinan dari *volumes* yang terdapat pada *instances*. Pengguna dapat membuat *instance* baru dari *snapshot* yang sudah dibuat dan dapat menghapus *snapshot*. *Snapshots* yang sudah dibuat memiliki konfigurasi dan sudah berisi data-data terakhir yang ada pada *instances*. Antarmuka *snapshots* dapat dilihat pada Gambar 4.7 Antarmuka *snapshots*.



Gambar 4.7 Antarmuka snapshots

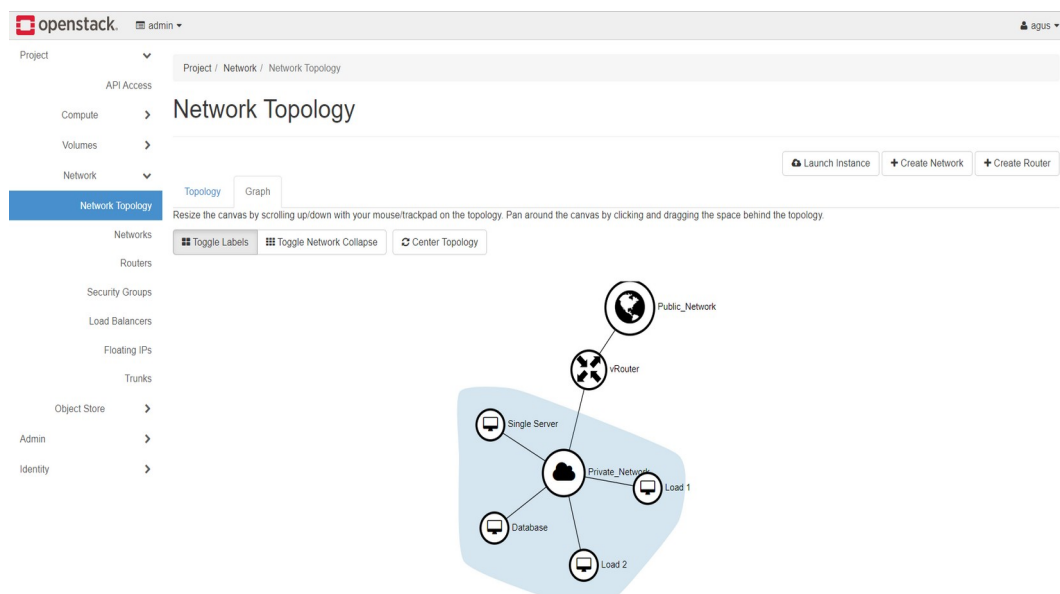
4.2.3 Antarmuka Network

Antarmuka *network* openstack merupakan antarmuka yang digunakan untuk mengelola layanan jaringan yang terdapat pada openstack. Antarmuka

network terdiri dari *network topology*, *networks*, *routers*, *security groups*, *load balancer* dan *floating ips*.

4.2.3.1 Antarmuka Network Topology

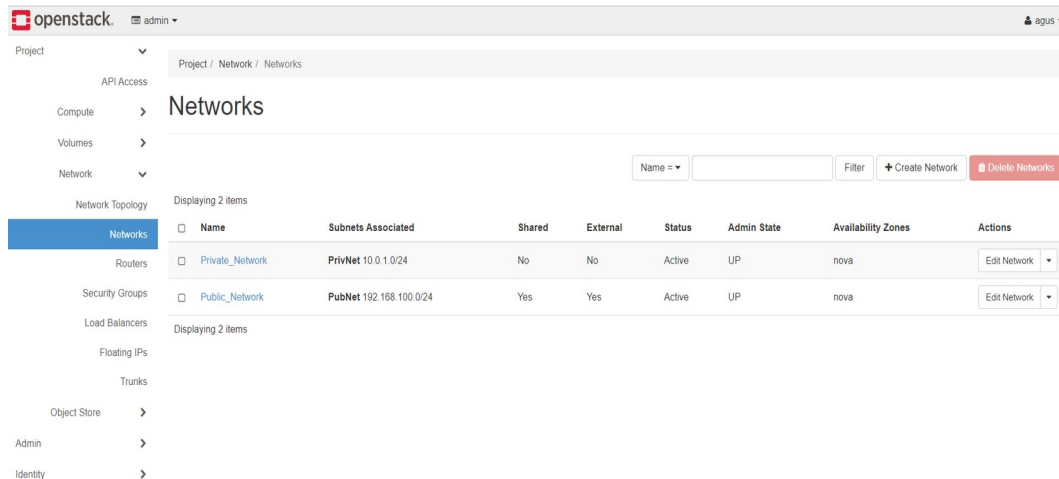
Antarmuka *network topology* openstack menampilkan topologi jaringan pada sistem yang sudah dibuat. Pada penelitian ini menggunakan satu jaringan publik, satu *virtual router*, dan satu jaringan privat dan empat *instances*. Antarmuka *network topologi* dapat dilihat pada Gambar 4.8 Antarmuka *network topology*.



Gambar 4.8 Antarmuka *network topology*

4.2.3.2 Antarmuka Networks

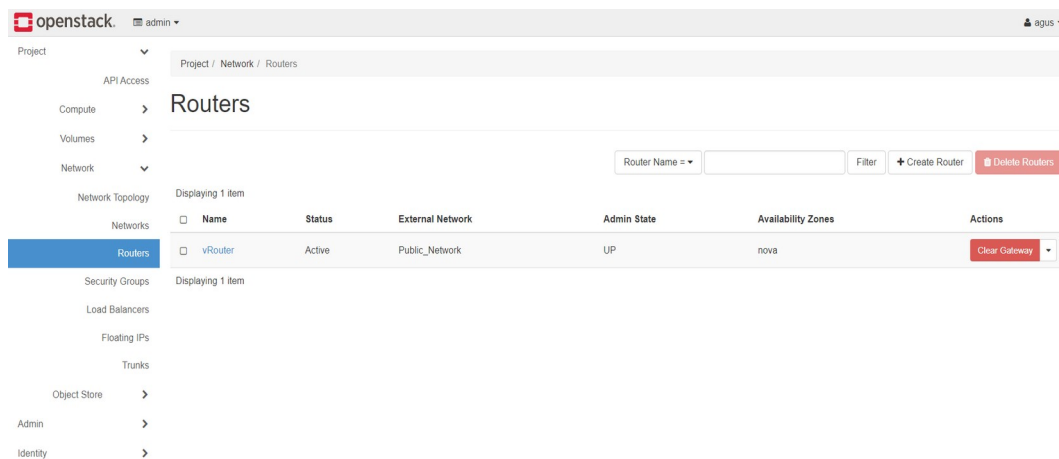
Antarmuka *networks* openstack menampilkan dan mengelola jaringan pada sistem openstack. *Networks* merupakan layanan yang digunakan untuk menghubungkan antara jaringan dari luar sistem openstack ke dalam sistem openstack. Dalam penelitian ini menggunakan dua jaringan yaitu, jaringan publik dan jaringan privat. Jaringan publik berfungsi untuk menghubungkan jaringan dari luar ke dalam ataupun sebaliknya, sedangkan jaringan privat berfungsi untuk memberikan alamat ip untuk tiap sistem seperti *instance* dan *virtual route*. Jaringan publik dan jaringan privat pada openstack dihubungkan menggunakan *floating ips*. Antarmuka *networks* dapat dilihat pada Gambar 4.9 Antarmuka *networks*.



Gambar 4.9 Antarmuka *networks*

4.2.3.3 Antarmuka *Routers*

Antarmuka *routers* openstack menampilkan dan mengelola *routers*. *Routers* berfungsi untuk mendistribusikan *ip address* secara statis ataupun dinamis. Dalam penelitian ini menggunakan satu *router* virtual yang terhubung ke empat *instance* dengan jaringan privat. Antarmuka *routers* dapat dilihat pada Gambar 4.10 Antarmuka *routers*.

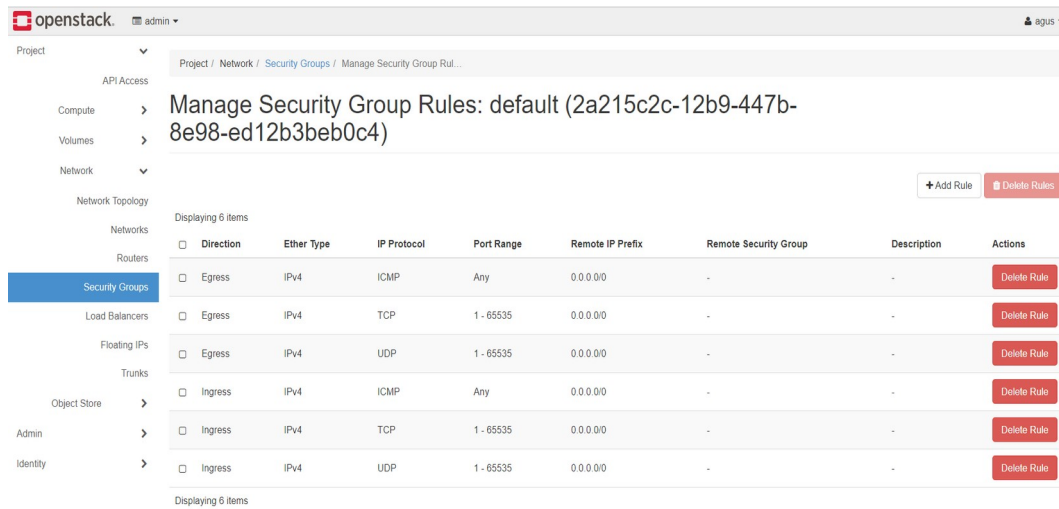


Gambar 4.10 Antarmuka *routers*

4.2.3.4 Antarmuka *Security Group*

Antarmuka *security group* openstack menampilkan dan mengelola *security group*. *Security group* merupakan layanan yang digunakan untuk mengatur port agar *instance* pada openstack bisa diakses secara keseluruhan atau terbatas dari dalam ataupun dari luar jaringan. Pada penelitian ini, seluruh port tcp dan udp

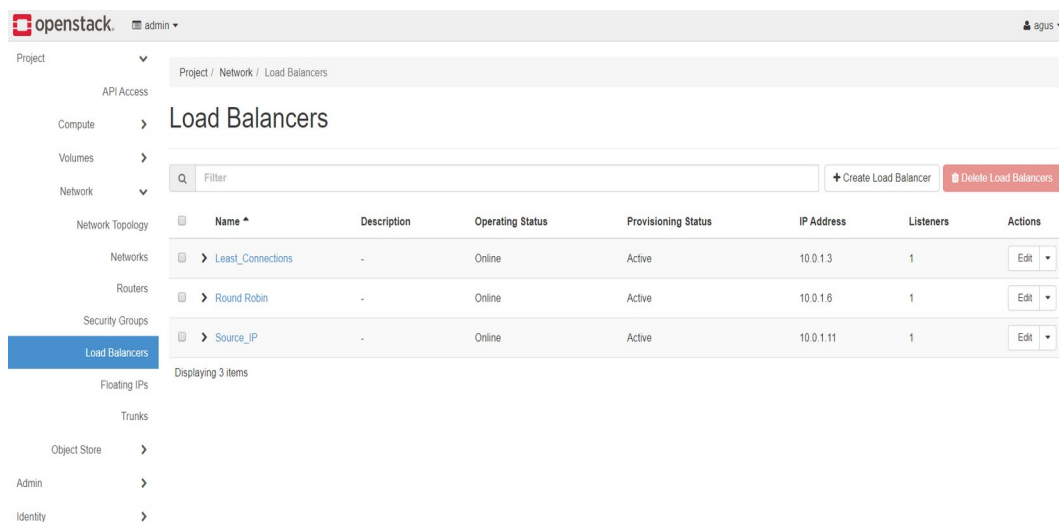
dibuka agar pengguna bisa mengakses jaringan. Antarmuka *security group* dapat dilihat pada Gambar 4.11 Antarmuka *security group*.



Gambar 4.11 Antarmuka *security group*

4.2.3.5 Antarmuka *Load Balancers*

Antarmuka *load balancers* openstack menampilkan dan mengelola *load balancer* yang akan digunakan untuk manajemen *web server* pada *instance* untuk meningkatkan performa *web server* pada *instance* dan untuk mengurangi terjadinya *fail transactions*. Dalam penelitian ini dibangun tiga jenis *load balancer* yaitu *round robin*, *least connections* dan *source ip*. Antarmuka *load balancer* dapat dilihat pada Gambar 4.12 Antarmuka *load balancer*.



Gambar 4.12 Antarmuka *load balancer*

4.2.3.6 Antarmuka *Floating IPs*

Antarmuka *floating ips* menampilkan dan mengelola *floating ips* yang digunakan untuk memberikan *ip address* pada *instance* agar bisa terhubung dengan mikrotik. *Floating ips* merupakan penghubung antara *ip address private* dengan *ip address* publik sehingga *instances* dapat diakses dari luar menggunakan *ip* publik yang sudah di hubungkan dengan *ip* lokal pada mikrotik. Antarmuka *floating ips* dapat dilihat pada Gambar 4.13 Antarmuka *floating ips*.

IP Address	Description	Mapped Fixed IP Address	Pool	Status	Actions
192.168.100.16		Single Server 10.0.1.13	Public_Network	Active	Disassociate
192.168.100.23		Load Balancer VIP 10.0.1.11	Public_Network	Active	Disassociate
192.168.100.17		Load 1 10.0.1.23	Public_Network	Active	Disassociate
192.168.100.12		Database 10.0.1.20	Public_Network	Active	Disassociate
192.168.100.18		Load 2 10.0.1.21	Public_Network	Active	Disassociate
192.168.100.22		Load Balancer VIP 10.0.1.3	Public_Network	Active	Disassociate
192.168.100.21		Load Balancer VIP 10.0.1.6	Public_Network	Active	Disassociate

Gambar 4.13 Antarmuka *floating ips*

4.3 Hasil Pengujian *Availability*

Pengujian *availability* dilakukan dengan cara menguji tingkat ketersediaan layanan terhadap *web server* pada *load server*. Pengujian ini dilakukan untuk mengetahui performa *availability* antar *load server*, setiap *server* akan ditentukan nilai *downtime* dan *uptimenya* sehingga akan diperoleh hasil nilai *availability*. Pengujian *availability* sangat berpengaruh terhadap jumlah waktu *downtime* yang dialami oleh *web server*. Pengujian *downtime* dilakukan untuk mengukur waktu yang diperlukan oleh *web server* untuk mengembalikan layanannya ketika diberikan gangguan. Pengujian *downtime* dilakukan menggunakan *tools siege* dengan cara memberikan *request* yang ditentukan dan jumlah pengulangan untuk setiap pengujian sebanyak sepuluh kali, sehingga akan didapat nilai jumlah *request* yang gagal ditangani *web server* dibagi jumlah rata-rata eksekusi *request* perdetik. Pengujian *availability* dilakukan pada tiga skenario

dan tiga kondisi jaringan yang berbeda yaitu jaringan lokal, jaringan sepi dan jaringan sibuk.

4.3.1 Hasil Pengujian *Availability* Jaringan Lokal

Pengujian *availability* dilakukan dengan cara mengamati kinerja *web server* pada saat diberikan beban dengan jumlah yang ditentukan. Pengujian dilakukan menggunakan jaringan lokal dengan memberikan gangguan kepada *web server* sesuai dengan skenario yang telah ditetapkan agar didapatkan nilai *downtime*. Pengujian pada jaringan lokal yaitu melakukan pengujian terhadap *web server* pada kondisi *server* tidak memiliki akses jaringan. Hasil pengujian yang diperoleh melalui *tools* *siege* yang terdapat pada lampiran A-1 ditampilkan pada tabel 4.1 berikut.

Tabel 4.1 Hasil Pengujian *Downtime* Jaringan Lokal

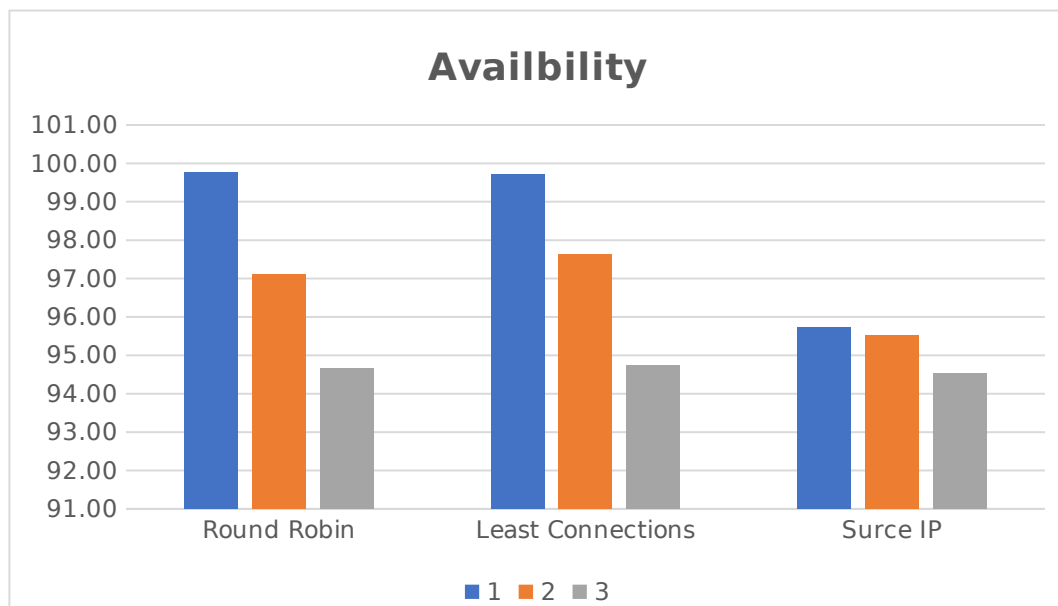
Server	Skenario	Downtime (Second)									
		ke-1	ke-2	ke-3	ke-4	ke-5	ke-6	ke-7	ke-8	ke-9	ke-10
Round Robin	1	0.27	0.00	0.00	0.03	0.13	0.03	0.02	0.07	0.23	0.00
	2	0.91	0.00	3.36	0.00	0.00	2.73	2.02	0.23	0.17	0.59
	3	1.49	0.00	1.73	0.05	6.80	1.15	1.62	0.00	5.74	0.00
Least Connections	1	0.38	0.08	0.00	0.07	0.29	0.05	0.06	0.00	0.00	0.02
	2	0.00	0.18	0.00	0.71	0.66	0.32	0.00	1.17	3.19	1.93
	3	2.12	0.31	1.48	1.47	0.32	2.70	5.61	2.75	0.00	2.32
Source IP	1	5.49	1.11	0.44	1.43	0.00	1.96	2.12	2.71	0.00	0.00
	2	1.51	2.07	0.71	4.10	0.87	0.00	3.07	0.70	1.19	1.50
	3	3.08	1.30	0.71	1.55	2.03	1.37	0.00	4.00	4.30	0.88

Berdasarkan hasil pengujian *downtime* yang ditampilkan pada Tabel 4.1, maka akan dilakukan perhitungan melalui persamaan (3.3) untuk menentukan nilai *availability* dari tiap *server*. Hasil perhitungan *availability* ditampilkan pada tabel 4.2 berikut.

Tabel 4.2 Hasil Perhitungan *Availability* Jaringan Lokal

Server	Skenario	Availability (%)										Rata-rata
		ke-1	ke-2	ke-3	ke-4	ke-5	ke-6	ke-7	ke-8	ke-9	ke-10	
Round Robin	1	99.14	100.00	100.00	99.90	99.60	99.90	99.95	99.80	99.30	100.00	99.76
	2	97.28	100.00	90.77	100.00	100.00	92.24	93.96	99.30	99.45	98.22	97.12
	3	95.56	100.00	94.90	99.85	81.16	96.43	95.23	100.00	83.52	100.00	94.67
Least Connections	1	98.84	99.75	100.00	99.80	99.09	99.85	99.80	100.00	100.00	99.95	99.71
	2	100.00	99.45	100.00	97.80	97.91	98.99	100.00	96.48	91.25	94.46	97.63
	3	93.84	98.99	95.62	95.62	98.99	92.18	86.88	92.24	100.00	92.99	94.73
Source IP	1	85.71	96.59	98.63	95.78	100.00	94.35	93.95	92.24	100.00	100.00	95.73
	2	95.78	93.90	97.80	89.01	97.23	100.00	91.54	97.75	96.37	95.67	95.51
	3	90.89	96.05	97.80	95.51	94.07	95.89	100.00	89.38	88.33	97.28	94.52

Berdasarkan Tabel 4.2 hasil perhitungan *availability*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *availability* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.14 grafik hasil pengujian *availability* jaringan lokal berikut ini.



Gambar 4.14 Hasil pengujian *availability* jaringan lokal

Berdasarkan hasil pengujian *availability* jaringan lokal pada skenario 1, *web server round robin* menunjukkan nilai tertinggi yaitu sebesar 99,76%, memiliki selisih 0,05% dari *least connections* dan nilai *availability* terendah

terdapat pada *web server source ip* yaitu sebesar 95,73%. Pada skenario 2, nilai *availability* tertinggi terdapat pada hasil pengujian *web server least connectios* sebesar 97,63% dengan selisih 0,51% dari *round robin* dan nilai *availbility* pada *source ip* memiliki nilai terendah yaitu sebesar 95,51%. Pada skenario 3 nilai *availability* tertinggi terdapat pada hasil pengujian *web server least connections* dengan nilai sebesar 94,73%, serta memiliki selisih nilai yang kecil sebesar 0,06% dari *round robin* dan nilai *availability* terendah terdapat pada *web server source ip* yaitu sebesar 94,52%.

4.3.2 Hasil Pengujian Availability Jaringan Sepi

Pengujian *availability* dilakukan dengan cara mengamati kinerja *web server* pada saat diberikan beban dengan jumlah yang ditentukan. Pengujian dilakukan menggunakan jaringan sepi dengan memberikan gangguan kepada *web server* sesuai dengan skenario yang telah ditetapkan agar didapatkan nilai *downtime*. Pengujian pada jaringan sepi yaitu melakukan pengujian terhadap *web server* pada jaringan untan2018 tidak banyak diakses oleh pengguna jam 02:00 – 04:00 dini hari dan jaringan berada pada trafik yang stabil. Hasil pengujian yang diperoleh melalui *tools siege* yang terdapat pada lampiran A-2 ditampilkan pada tabel 4.3 berikut.

Tabel 4.3 Hasil Pengujian *Downtime* Jaringan Sepi

Server	Skenario	Downtime (Second)									
		ke-1	ke-2	ke-3	ke-4	ke-5	ke-6	ke-7	ke-8	ke-9	ke-10
Round Robin	1	0.06	0.45	1.66	0.39	0.11	0.40	0.42	0.12	0.46	2.77
	2	3.06	1.01	2.46	2.78	3.94	5.70	2.93	3.40	0.73	2.90
	3	5.36	1.09	2.20	3.08	2.57	4.25	2.09	5.60	2.26	4.79
Least Connections	1	1.44	0.10	1.69	0.49	0.49	0.48	1.66	0.49	4.00	0.11
	2	2.71	2.06	1.31	0.32	1.89	0.00	3.16	0.62	2.57	2.65
	3	5.87	0.34	2.47	0.42	0.82	1.25	3.73	2.05	2.16	6.01
Source IP	1	4.90	1.19	1.75	2.49	3.15	2.02	3.17	1.46	0.01	2.33
	2	3.30	2.66	3.15	3.22	0.17	0.97	2.61	3.43	0.86	3.29
	3	3.79	2.16	1.83	2.02	0.00	16.10	0.31	1.36	3.55	0.89

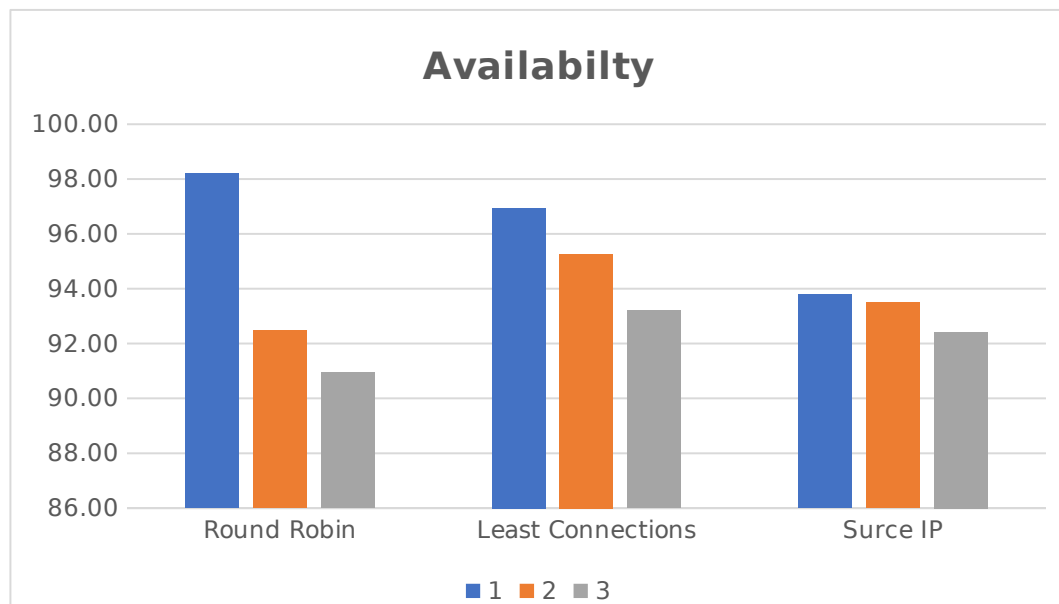
Berdasarkan hasil pengujian *downtime* yang ditampilkan pada Tabel 4.3, maka akan dilakukan perhitungan melalui persamaan (3.3) untuk menentukan

nilai *availability* dari tiap *server*. Hasil perhitungan *availability* ditampilkan pada tabel 4.4 berikut.

Tabel 4.4 Hasil Perhitungan *Availability* Jaringan Sepi

Server	Skenario	Availability (%)										Rata-rata
		ke-1	ke-2	ke-3	ke-4	ke-5	ke-6	ke-7	ke-8	ke-9	ke-10	
Round Robin	1	99.85	98.68	95.45	98.84	99.65	98.73	98.89	99.70	98.84	93.45	98.21
	2	92.36	97.17	93.33	92.47	90.41	86.17	91.95	90.83	97.80	92.18	92.47
	3	85.84	96.85	93.90	91.54	92.93	88.83	94.29	85.19	92.65	87.45	90.95
Least Connections	1	96.10	99.75	94.96	98.63	98.68	98.53	95.51	98.68	89.01	99.65	96.95
	2	92.59	94.24	96.05	99.09	94.29	100.00	91.60	98.12	94.18	92.53	95.27
	3	85.06	98.94	92.76	98.63	97.91	96.37	90.89	94.63	94.40	82.35	93.19
Source IP	1	86.82	96.59	95.18	93.10	91.36	94.29	91.19	95.94	99.95	93.33	93.78
	2	90.71	92.53	91.78	91.48	99.55	97.12	92.53	91.07	97.54	90.77	93.51
	3	89.75	93.79	94.46	94.18	100.00	68.33	99.04	96.32	90.83	97.49	92.42

Berdasarkan Tabel 4.4 hasil perhitungan *availability*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *availability* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.15 grafik hasil pengujian *availability* jaringan lokal berikut ini.



Gambar 4. 15 Hasil pengujian *availability* jaringan sepi

Berdasarkan hasil pengujian *availability* jaringan sepi pada skenario 1 *web server round robin* menunjukkan hasil tertinggi dengan nilai sebesar 98,21% serta memiliki selisih 1,26% dari *web server least connections* dan nilai *availability* terendah terdapat pada *web server source ip* sebesar 93,78%. Pada skenario 2 nilai *availability* tertinggi terdapat pada hasil pengujian *web server least connections* dengan nilai sebesar 95,27% memiliki selisih 1,76% dari *web server source ip* dan nilai *availability* terendah terdapat pada *web server round robin* sebesar 92,47%. Pada skenario 3 nilai *availability* tertinggi terdapat pada hasil pengujian *web server least connections* dengan nilai sebesar 93,19% memiliki selisih nilai 0,77% dari *web server source ip* dan nilai *availability web server round robin* memiliki nilai dibawah *server least connectons* dan *source ip* dengan nilai sebesar 90,95%.

4.3.3 Hasil Pengujian Availability Jaringan Sibuk

Pengujian *availability* dilakukan dengan cara mengamati kinerja *web server* pada saat diberikan beban dengan jumlah yang ditentukan. Pengujian dilakukan menggunakan jaringan sibuk dengan memberikan gangguan kepada *web server* sesuai dengan skenario yang telah ditetapkan agar didapatkan nilai *downtime*. Pengujian pada jaringan sibuk yaitu melakukan pengujian terhadap *web server* pada saat jaringan untan2018 diakses oleh banyak pengguna jam 10:00 – 12:00 dan jaringan berada pada trafik yang tidak stabil. Hasil pengujian yang diperoleh melalui *tools siege* yang terdapat pada lampiran A-3 ditampilkan pada tabel 4.5 berikut.

Tabel 4.5 Hasil Pengujian Downtime Jaringan Sibuk

Server	Skenario	Downtime (Second)									
		ke-1	ke-2	ke-3	ke-4	ke-5	ke-6	ke-7	ke-8	ke-9	ke-10
Round Robin	1	2.74	3.03	2.07	2.46	2.78	5.25	0.58	9.89	1.20	1.38
	2	1.62	2.95	2.91	2.29	1.20	3.58	0.82	6.18	10.32	3.75
	3	5.45	2.66	7.26	2.73	5.61	0.73	3.53	2.88	3.91	1.96
Least Connections	1	3.28	1.83	0.11	3.64	6.07	1.37	1.32	5.34	2.39	2.00
	2	3.39	8.51	5.50	4.58	0.09	5.03	0.40	0.62	1.82	5.59
	3	5.96	1.33	0.43	5.40	6.44	3.92	3.21	2.03	5.57	4.12
Source IP	1	4.84	3.48	1.24	4.26	9.12	2.07	0.78	3.39	1.74	3.83

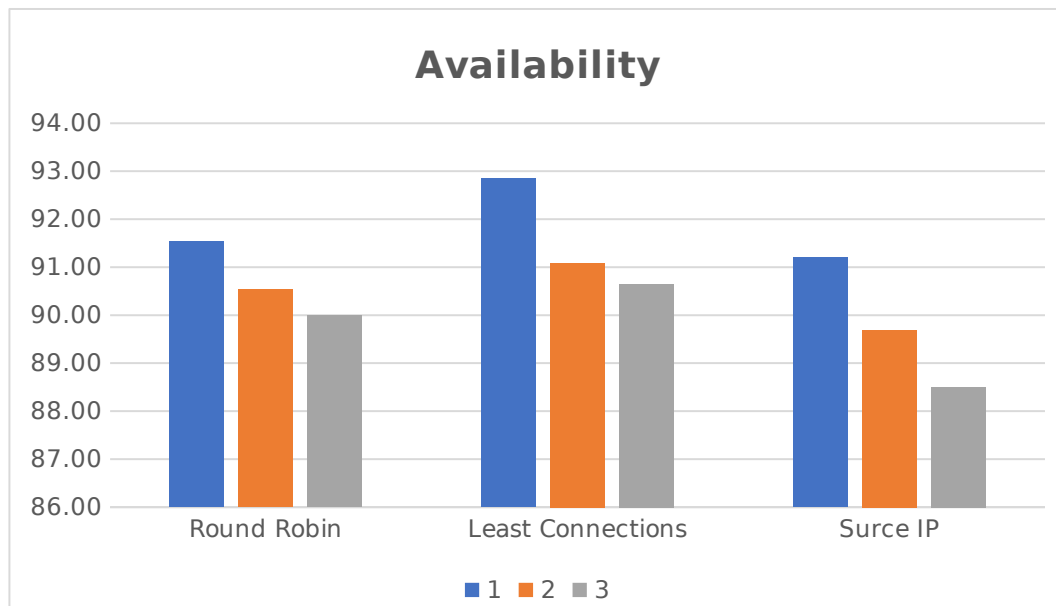
Server	Skenario	Downtime (Second)									
		ke-1	ke-2	ke-3	ke-4	ke-5	ke-6	ke-7	ke-8	ke-9	ke-10
	2	3.30	1.81	8.92	2.55	4.43	1.90	3.10	3.38	5.63	4.55
	3	1.82	3.46	2.03	10.89	3.97	4.17	3.11	4.38	4.37	5.24

Berdasarkan hasil pengujian *downtime* yang ditampilkan pada Tabel 4.5, maka akan dilakukan perhitungan melalui persamaan (3.3) untuk menentukan nilai *availability* dari tiap *server*. Hasil perhitungan *availability* ditampilkan pada tabel 4.6 berikut.

Tabel 4.6 Hasil Perhitungan *Availability* Jaringan Sibuk

Server	Skenario	Availability (%)										Rata-rata
		ke-1	ke-2	ke-3	ke-4	ke-5	ke-6	ke-7	ke-8	ke-9	ke-10	
Round Robin	1	92.82	91.83	93.90	93.33	92.70	85.1 2	98.48	74.61	96.32	96.21	91.53
	2	95.12	91.78	91.83	93.45	96.96	90.4 7	97.33	83.11	73.82	91.42	90.53
	3	85.52	92.24	81.16	92.70	85.12	98.0 6	90.23	91.30	89.01	94.51	89.99
Least Connections	1	91.72	95.07	99.70	89.69	83.92	96.4 3	96.64	86.81	93.16	95.34	92.85
	2	91.13	80.31	86.24	86.36	99.75	86.3 6	99.09	98.27	96.10	87.26	91.09
	3	84.26	96.59	98.89	86.23	83.79	89.6 9	92.99	95.07	85.06	93.96	90.65
Source IP	1	87.58	91.07	96.64	89.50	79.52	94.4 0	97.86	91.36	95.18	88.89	91.20
	2	91.31	95.23	74.36	94.11	88.91	95.8 6	91.50	92.91	84.53	88.20	89.69
	3	94.79	89.93	94.57	71.79	89.99	89.8 7	90.71	87.77	89.20	86.43	88.50

Berdasarkan Tabel 4.6 hasil perhitungan *availability*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *availability* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.16 grafik hasil pengujian *availability* jaringan sibuk berikut ini.



Gambar 4. 16 Hasil pengujian *availability* jaringan sibuk

Berdasarkan hasil pegujian *availability* jaringan sibuk pada skenario 1, *web server least connection* memiliki nilai tertinggi sebesar 92,85% serta memiliki selisih yang kecil terhadap *web server round robin* yaitu sebesar 1,32% dan nilai *availability* terendah terdapat pada *web server source ip* yaitu sebesar 91.20%. pada skenario 2, nilai *availability* tertinggi terdapat pada hasil pengujian *web server least connections* dengan nilai sebesar 91,09% memiliki selisih nilai yang kecil terhadap *web server roud robin* sebesar 0,56% dan selisih nilai 1,4% dari *web server source ip*. Pada skenario 3, nilai *availability* tertinggi terdapat pada hasil pengujian *web server least connections* sebesar 90,65% dan nilai *availability web server round robin* sebesar 89,99% berada dibawah nilai *web server least connections* dan memiliki selisih 1,49% dari *web server source ip*.

4.3.4 Analisis Hasil Pengujian *Availability*

Berdasarkan hasil pengujian *availability* didapatkan hasil ketika *load server* mengalami *down* pada saat pengujian, *web server* masih bisa diakses oleh *client* hal ini karena layanan *high availability web server* tersebut berjalan dengan baik meskipun terdapat *down*. *Web server* yang memiliki *load balancer* bisa diakses oleh *client* karena apabila terjadi *down* pada salah satu *server* maka *server*

yang lain akan menangani permintaan *request* dari *client* sehingga *web server* masih bisa diakses.

Hasil pengujian *availability* yang dilakukan pada tiga kondisi jaringan yang berbeda memiliki nilai *availability* yang bervariasi terhadap tiga skenario yang diterapkan. Hasil pengujian *availability* tertinggi pada skenario pertama pada jaringan lokal dan sepi terdapat pada *web server round robin* yaitu sebesar 99,76% dan 98,21%, hal ini dikarenakan algoritma *round robin* bekerja sesuai dengan pembagian jumlah *request* yang sama rata terhadap jumlah *web server* tersedia, sehingga algoritma ini dapat bekerja secara maksimal dengan kondisi *server* tidak terjadi *down* dan dalam jaringan yang memiliki trafik yang rendah. Hasil pengujian tertinggi pada skenario kedua dan ketiga dengan kondisi jaringan lokal, jaringan sepi dan jaringan sibuk terdapat pada hasil pengujian *web server least connections*, hal ini dikarenakan algoritma *least connections* bekerja dengan maksimal ketika *server* terjadi *down*, maka algoritma *least connections* akan memprioritaskan *server* atau memilih jalur koneksi yang paling sedikit yang akan diakses oleh *client* sehingga mengurangi terjadinya *web server* gagal diakses oleh *client*. Pada pengujian *web server source ip* memiliki nilai *availability* dibawah *server round robin* dan *server least connections* hal ini karena algoritma *source ip* bekerja berdasarkan ip unik yang dimiliki oleh *client* dan diteruskan ke *server*.

Berdasarkan hasil pengujian *availability* jika ditinjau dari pengujian parameter *failed transaction* yang terdapat pada lampiran A-1, maka skenario 1 pada algoritma *round robin* dan *least connections* memiliki nilai tidak mengalami kenaikan dan penurunan yang signifikan, berbeda dengan algoritma *source ip* yang memiliki nilai *failed transaction* yang memiliki perbedaan yang signifikan, hal ini disebabkan ketika dilakukan pengujian kondisi *web server* tidak bisa bekerja secara maksimal apabila *web server* dalam kondisi terus menerus dilakukan pengujian. Pada skenario 2 dan skenario 3 pada setiap algoritma memiliki nilai *failed connections* mengalami kenaikan dan penurunan yang signifikan dari setiap pengulangan pengujian, hal ini disebabkan ketika dilakukan pengujian pada *web server* yang menangani jumlah *request* yang dikirim terus-menerus tanpa henti dari setiap pengulangan pengujian *web server* dipaksa

bekerja secara optimal sedangkan jumlah *request* yang dikirim tidak sebanding dengan jumlah *web server* yang tersedia, sehingga mengakibatkan nilai *failed connections* tidak stabil. Pengujian parameter *failed connectios* pada metode pengujian *availability* sangat berpengaruh terhadap kondisi jaringan, ketersediaan *web server* dan kondisi ketersediaan *cpu* saat dilakukan pengujian, sehingga mempengaruhi hasil dari metode pengujian *availability*, apabila nilai *failed connections* yang diperoleh dari setiap pengujian itu rendah maka hasil *availability* akan tinggi, apabila nilai *failed connections* tinggi maka hasil dari metode pengujian *availability*nya akan rendah.

4.4 Hasil Pengujian *Quality of Service*

Pengujian *quality of service* dilakukan dengan cara memberikan beban terhadap *web server* dengan jumlah yang sudah ditentukan dan dilakukan pengulangan sebanyak sepuluh kali. Pengujian ini bertujuan untuk mengetahui performa dari *web server* yang menggunakan metode *load balancing*. Parameter dalam pengujian ini yaitu *throughput*, *delay*, *jitter* dan *packet loss*. Hasil pengujian tersebut akan menentukan nilai *quality of service*. Pengujian *quality of service* dilakukan dengan kondisi *server* normal tanpa ada yang *down* dan dilakukan pada tiga kondisi jaringan yaitu jaringan lokal, jaringan sepi dan jaringan sibuk. Adapun pengujian dilakukan sebagai berikut.

4.4.1 Hasil Pengujian *Quality of Service* Jaringan Lokal

Pengujian *quality of service* dilakukan dengan cara mengamati kinerja *web server* pada saat diberikan beban dengan jumlah yang ditentukan. Pengujian dilakukan menggunakan jaringan lokal dengan memberikan gangguan kepada *web server* untuk mendapatkan nilai dari parameter *throughput*, *delay*, *jitter* dan *packet loss*. Pengujian ini dilakukan dengan cara mengirimkan *request* sebesar 100, 200, 300, hingga 1500 *request* terhadap *web server* pada *load server*. Hasil pengujian yang diperoleh melalui *tools siege*. Adapun hasil pengujian yang didapat sebagai berikut.

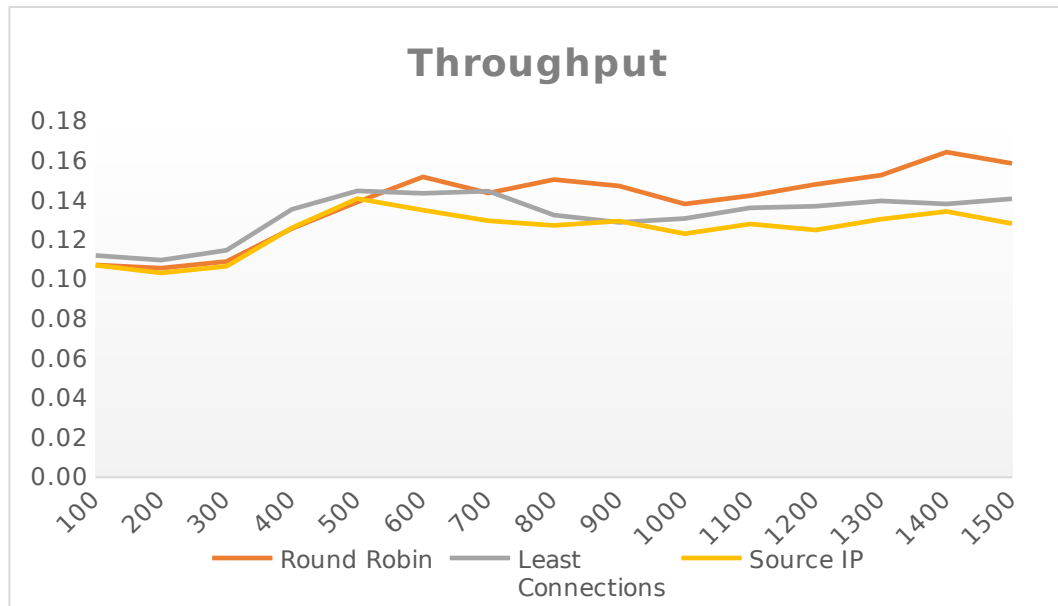
4.4.1.1 Pengujian *Throughput*

Pengujian parameter *throughput* dilakukan untuk mengetahui kemampuan *web server* dalam memberikan layanan secara benar terhadap *request* yang datang secara bersamaan. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan pada waktu yang diperlukan oleh *web server* dalam menangani *request* dan jumlah *bandwidth* yang diperlukan selama pengujian, sehingga dari hasil pengamatan akan dilakukan perhitungan *throughput* dari setiap kali pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-1 dan dilakukan perhitungan *throughput* menggunakan persamaan (3.4). Hasil perhitungan *throughput* dapat dilihat pada tabel 4.7 berikut.

Tabel 4.7 Hasil Perhitungan *Throughput* Jaringan Lokal

<i>Request</i>	<i>Throughput (Mbps)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	0.11	0.11	0.11
200	0.11	0.11	0.10
300	0.11	0.11	0.11
400	0.13	0.14	0.13
500	0.14	0.14	0.14
600	0.15	0.14	0.14
700	0.14	0.14	0.13
800	0.15	0.13	0.13
900	0.15	0.13	0.13
1000	0.14	0.13	0.12
1100	0.14	0.14	0.13
1200	0.15	0.14	0.13
1300	0.15	0.14	0.13
1400	0.16	0.14	0.13
1500	0.16	0.14	0.13
Rata-rata	0.14	0.13	0.13

Berdasarkan Tabel 4.7 hasil perhitungan *throughput*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *throughput* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.17 grafik hasil pengujian *throughput* jaringan lokal berikut ini.



Gambar 4.17 Hasil pengujian *throughput* jaringan lokal

Berdasarkan hasil pengujian parameter *throughput* pada jaringan lokal, *web server round robin* memiliki rata-rata nilai *throughput* tertinggi sebesar 0.14 Mbps dan memiliki grafik yang cenderung naik dari setiap kali dilakukan penambahan jumlah *request*. Nilai *throughput* pada *web server least connections* dan *source ip* memiliki rata-rata nilai yang sama yaitu sebesar 0.13 Mbps. Grafik *throughput* pada *web server least connections* dan *source ip* memiliki grafik yang cenderung stabil.

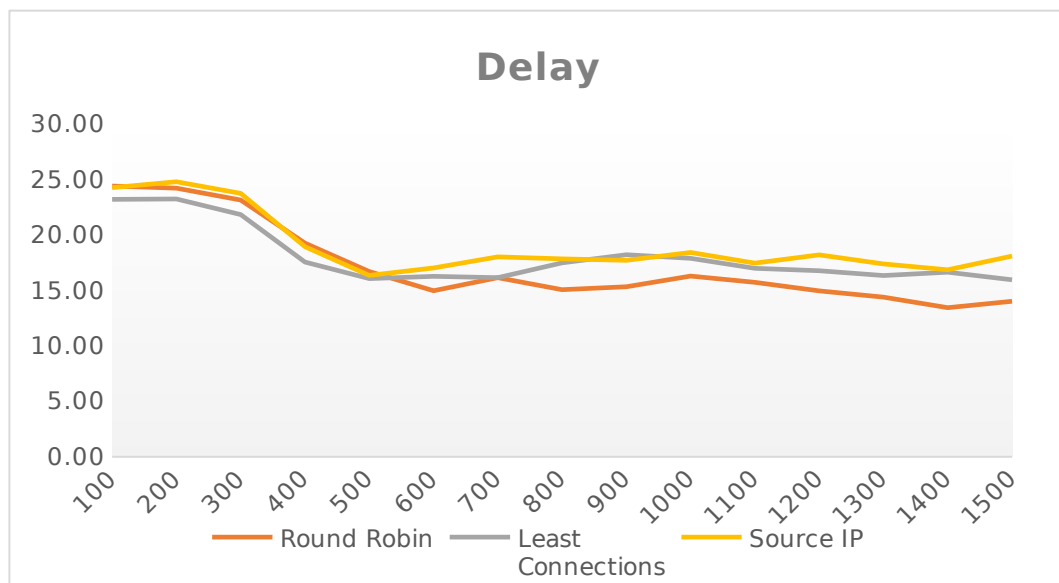
4.4.1.2 Pengujian Delay

Pengujian parameter *delay (legacy)* dilakukan untuk mengetahui waktu yang diperlukan untuk mengirim sejumlah *request* ke *web server*. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan jumlah waktu yang diperlukan selama pengiriman *request* dan jumlah *request* yang dapat diterima oleh *web server* selama pengujian, sehingga dari hasil pengamatan akan dilakukan perhitungan *delay* dari setiap pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-1 dan dilakukan perhitungan *delay* menggunakan persamaan (3.5). Hasil perhitungan *delay* dapat dilihat pada tabel 4.8 berikut.

Tabel 4.8 Hasil Perhitungan *Delay* Jaringan Lokal

<i>Request</i>	<i>Delay (ms)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	24.41	23.21	24.27
200	24.22	23.24	24.80
300	23.15	21.83	23.75
400	19.27	17.56	18.93
500	16.70	16.06	16.36
600	14.97	16.26	17.03
700	16.15	16.15	18.02
800	15.06	17.48	17.84
900	15.34	18.21	17.71
1000	16.29	17.89	18.42
1100	15.74	16.99	17.46
1200	14.96	16.78	18.21
1300	14.39	16.34	17.39
1400	13.44	16.63	16.86
1500	14.02	15.95	18.10
Rata-rata	17.21	18.04	19.01

Berdasarkan Tabel 4.8 hasil perhitungan *delay*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *delay* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.18 grafik hasil pengujian *delay* jaringan lokal berikut ini.



Gambar 4.18 Hasil pengujian *delay* jaringan lokal

Berdasarkan hasil pengujian parameter *delay* terhadap *web server* pada jaringan lokal, *web server round robin* memiliki nilai *delay* terendah yaitu sebesar 17.21 ms, serta memiliki selisih 0,83 ms dari *least connections* dan nilai *delay* tertinggi terdapat pada *web server source ip* yaitu sebesar 19,01 ms. Berdasarkan kategori latensi dari semua *web server* termasuk kategori sangat bagus karena rata-rata hasil pengujian *delay* dari setiap pengujian *load server* kurang dari 150 ms. *Web server round robin* memiliki grafik yang cenderung menurun ketika *request* yang dikirim semakin bertambah dan pada *web server least connections* dan *source ip* memiliki grafik yang cenderung stabil.

4.4.1.3 Pengujian Jitter

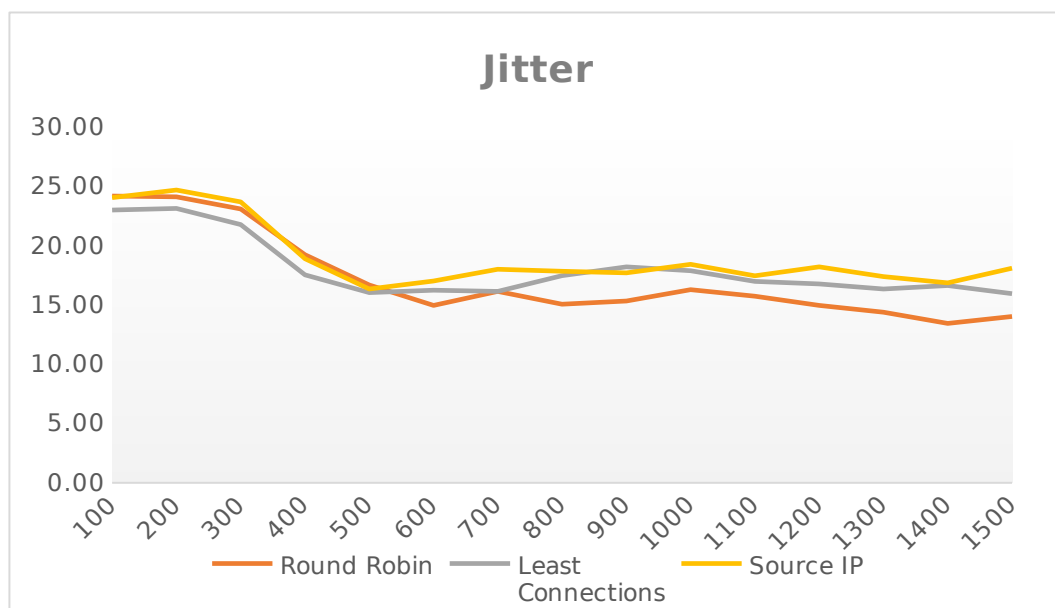
Pengujian parameter *jitter* memiliki kaitan erat dengan parameter *delay*. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan selisih antara *delay* pertama dengan *delay* selanjutnya, sehingga dari hasil pengamatan akan dilakukan perhitungan *jitter* dari setiap pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-1 dan dilakukan perhitungan *jitter* menggunakan persamaan (3.7). Hasil perhitungan *jitter* dapat dilihat pada tabel 4.9 berikut.

Tabel 4.9 Hasil Perhitungan *Jitter* Jaringan Lokal

<i>Request</i>	<i>Jitter (ms)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	24.17	22.98	24.03
200	24.09	23.12	24.67
300	23.08	21.76	23.67
400	19.22	17.52	18.88
500	16.66	16.03	16.33
600	14.95	16.23	17.00
700	16.13	16.12	17.99
800	15.04	17.45	17.82
900	15.32	18.19	17.69
1000	16.28	17.87	18.41
1100	15.72	16.97	17.45
1200	14.95	16.76	18.19
1300	14.38	16.33	17.37
1400	13.43	16.62	16.85
1500	14.01	15.94	18.09

Request	Jitter (ms)		
	Round Robin	Least Connections	Source IP
Rata-rata	17.16	17.99	18.96

Berdasarkan Tabel 4.9 hasil perhitungan *jitter*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *jitter* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.19 grafik hasil pengujian *jitter* jaringan lokal berikut ini.



Gambar 4.19 Hasil pengujian *jitter* jaringan lokal

Berdasarkan hasil pengujian parameter *jitter* pada jaringan lokal terhadap *web server*. Pada *web server round robin* memiliki nilai *jitter* yang terendah dari *web server* lain yaitu sebesar 17.16 ms dan memiliki selisih 0.83 ms dari *web server least connections*. Nilai *Jitter* tertinggi terdapat pada *web server source ip* yaitu sebesar 18.96 ms. Berdasarkan Kategori degradasi *jitter* dari semua *load server* yang dilakukan pengujian termasuk kategori bagus karena berada pada 0 hingga 75 ms. *Web server round robin* memiliki grafik yang cenderung menurun ketika *request* yang dikirim semakin bertambah dan pada *web server least connections* dan *source ip* memiliki grafik yang cenderung stabil.

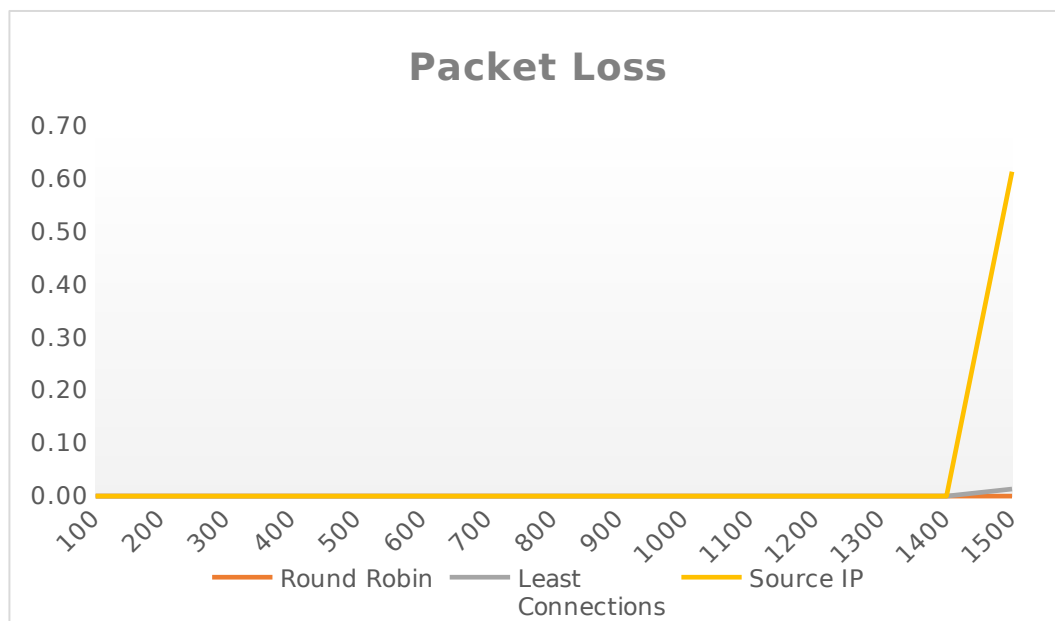
4.4.1.4 Pengujian *Packet Loss*

Pengujian parameter *packet loss* dilakukan untuk mengetahui jumlah *request* yang dapat ditangani oleh *web server*. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan *request* yang berhasil ditangani oleh *web server* setiap kali pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-1 dan dilakukan perhitungan nilai *packet loss* menggunakan persamaan (3.8). Hasil perhitungan *Packet loss* dapat dilihat pada tabel 4.10 berikut.

Tabel 4.10 Hasil Perhitungan *Packet Loss* Jaringan Lokal

<i>Request</i>	<i>Packet Loss (%)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	0.00	0.00	0.00
200	0.00	0.00	0.00
300	0.00	0.00	0.00
400	0.00	0.00	0.00
500	0.00	0.00	0.00
600	0.00	0.00	0.00
700	0.00	0.00	0.00
800	0.00	0.00	0.00
900	0.00	0.00	0.00
1000	0.00	0.00	0.00
1100	0.00	0.00	0.00
1200	0.00	0.00	0.00
1300	0.00	0.00	0.00
1400	0.00	0.00	0.00
1500	0.00	0.01	0.61
Rata-rata	0.00	0.00	0.04

Berdasarkan Tabel 4.10 hasil perhitungan *packet loss*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *packet loss* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.20 grafik hasil pengujian *packet loss* jaringan lokal berikut ini.



Gambar 4.20 Hasil pengujian *packet loss* jaringan lokal

Berdasarkan hasil pengujian parameter *packet loss* terhadap *web server* pada jaringan lokal, nilai *packet loss* terendah terdapat pada *web server round robin* dan *least connections* yaitu sebesar 0 %, sedangkan pada *web server source ip* memiliki nilai *packet loss* tertinggi yaitu sebesar 0.61%. Nilai *packet loss* pada *load server* termasuk kategori sangat bagus karena memiliki nilai *packet loss* yang berada pada 0 hingga 2% yang terdapat pada kategori degradasi *packet loss*. *Web server* pada *web server round robin* dan *least connections* memiliki grafik yang stabil dan tidak terjadi kegagalan transaksi dari 100 hingga 1500 *request* dan pada *web server source ip* memiliki grafik yang meningkat pada 1500 *request*.

4.4.2 Hasil Pengujian *Quality of Service* Jaringan Sepi

Pengujian *quality of service* dilakukan dengan cara mengamati kinerja *web server* pada saat diberikan beban dengan jumlah yang ditentukan. Pengujian dilakukan menggunakan jaringan sepi dengan memberikan gangguan terhadap *web server* untuk mendapatkan nilai dari parameter *throughput*, *delay*, *jitter* dan *packet loss*. Pengujian pada jaringan sepi yaitu melakukan pengujian terhadap *web server* pada jaringan untan2018 yang tidak banyak diakses oleh pengguna pada jam 02:00 – 04:00 dini hari dan jaringan berada pada trafik yang stabil. Hasil

pengujian yang diperoleh melalui *tools* siege. Adapun hasil pengujian yang diadapat sebagai berikut.

4.4.2.1 Pengujian *Throughput*

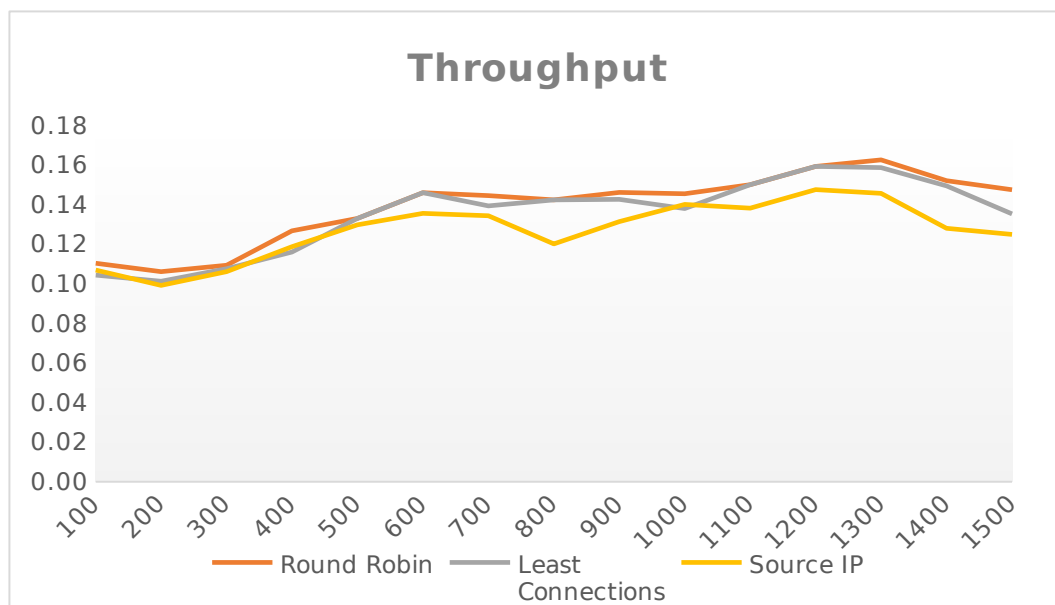
Pengujian parameter *throughput* dilakukan untuk mengetahui kemampuan *web server* dalam memberikan layanan secara benar terhadap *request* yang datang secara bersamaan. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan pada waktu yang diperlukan oleh *web server* dalam menangani *request* dan jumlah *bandwidth* yang diperlukan selama pengujian, sehingga dari hasil pengamatan akan dilakukan perhitungan *throughput* dari setiap kali pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-2 dan dilakukan perhitungan *throughput* menggaunakan persamaan (3.4). Hasil perhitungan *throughput* dapat dilihat pada tabel 4.11 berikut.

Tabel 4.11 Hasil Perhitungan *Throughput* Jaringan Sepi

<i>Request</i>	<i>Throughput (Mbps)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	0.11	0.10	0.11
200	0.11	0.10	0.10
300	0.11	0.11	0.11
400	0.13	0.12	0.12
500	0.13	0.13	0.13
600	0.15	0.15	0.14
700	0.14	0.14	0.13
800	0.14	0.14	0.12
900	0.15	0.14	0.13
1000	0.15	0.14	0.14
1100	0.15	0.15	0.14
1200	0.16	0.16	0.15
1300	0.16	0.16	0.15
1400	0.15	0.15	0.13
1500	0.15	0.14	0.13
Rata-rata	0.14	0.14	0.13

Berdasarkan Tabel 4.11 hasil perhitungan *throughput*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *throughput* dari setiap *server*

dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.21 grafik hasil pengujian *throughput* jaringan sepi berikut ini.



Gambar 4.21 Hasil pengujian *throughput* jaringan sepi

Berdasarkan hasil pengujian parameter *throughput* terhadap *web server* pada jaringan sepi, *web server round robin* dan *least connections* memiliki rata-rata nilai *throughput* tertinggi sebesar 0.14 Mbps. Kedua *web server* ini memiliki grafik *throughput* yang cenderung meningkat dan terjadi penurunan nilai *throughput* pada 1500 *request*. Nilai *throughput* pada *server source ip* berada dibawah *server round robin* dan *least connections* yaitu sebesar 0,13 Mbps serta memiliki grafik yang tidak stabil.

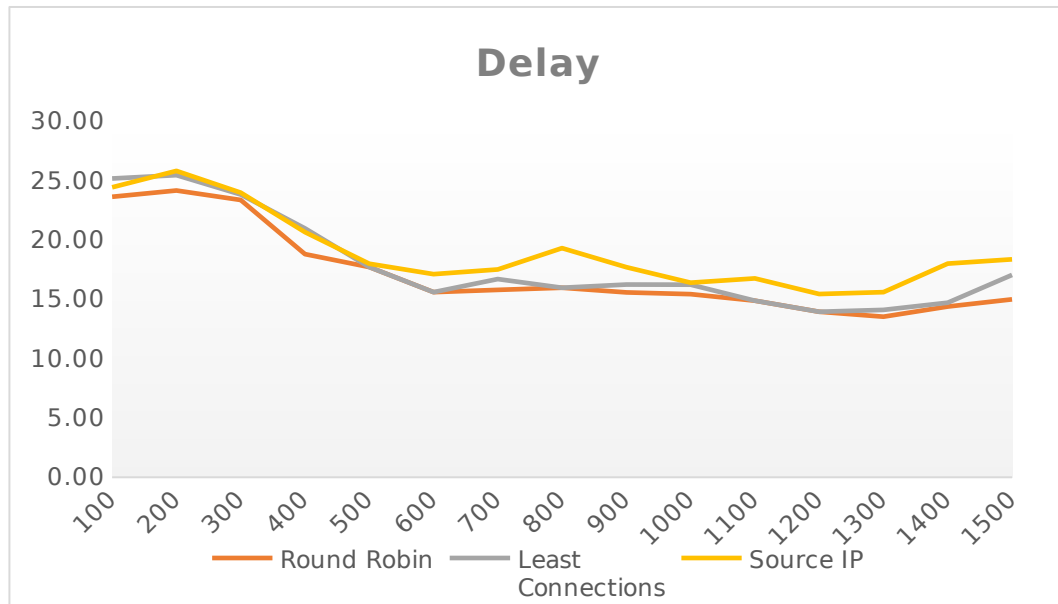
4.4.2.2 Pengujian Delay

Pengujian parameter *delay (legacy)* dilakukan untuk mengetahui waktu yang diperlukan untuk mengirim sejumlah *request* ke *web server*. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan jumlah waktu yang diperlukan selama pengiriman *request* dan jumlah *request* yang dapat diterima oleh *web server* selama pengujian, sehingga dari hasil pengamatan akan dilakukan perhitungan *delay* dari setiap pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-2 dan dilakukan perhitungan *delay* menggunakan persamaan (3.5). Hasil perhitungan *delay* dapat dilihat pada tabel 4.12 berikut.

Tabel 4.12 Hasil Perhitungan *Delay* Jaringan Sepi

<i>Request</i>	<i>Delay (ms)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	23.64	25.18	24.43
200	24.17	25.46	25.82
300	23.36	23.82	23.98
400	18.80	20.98	20.66
500	17.71	17.71	18.00
600	15.60	15.60	17.12
700	15.79	16.70	17.51
800	15.97	15.97	19.31
900	15.57	16.24	17.71
1000	15.43	16.23	16.40
1100	14.88	14.88	16.75
1200	13.94	13.94	15.45
1300	13.53	14.10	15.60
1400	14.38	14.71	18.01
1500	15.00	17.05	18.36
Rata-rata	17.19	17.90	19.01

Berdasarkan Tabel 4.12 hasil perhitungan *delay*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *delay* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.18 grafik hasil pengujian *delay* jaringan lokal berikut ini.



Gambar 4.22 Hasil pengujian *delay* jaringan sepi

Berdasarkan hasil pengujian parameter *delay* terhadap *web server* pada jaringan sepi, nilai *delay* terendah terdapat pada *web server round robin* yaitu sebesar 17.19 ms dan memiliki selisih 0.81 ms dari *least connections*. Kedua *web server* ini memiliki trafik *delay* yang cenderung menurun ketika jumlah *request* ditambah pada setiap kali pengujian. Nilai *delay* tertinggi terdapat pada *web server source ip* yaitu sebesar 19.01 ms. Pada *web server source ip* memiliki grafik yang tidak stabil dan cenderung menurun. Berdasarkan kategori latensi *delay*, nilai *delay load server* terdapat pada kategori sangat bagus, diakrenakan kurang dari 150 ms.

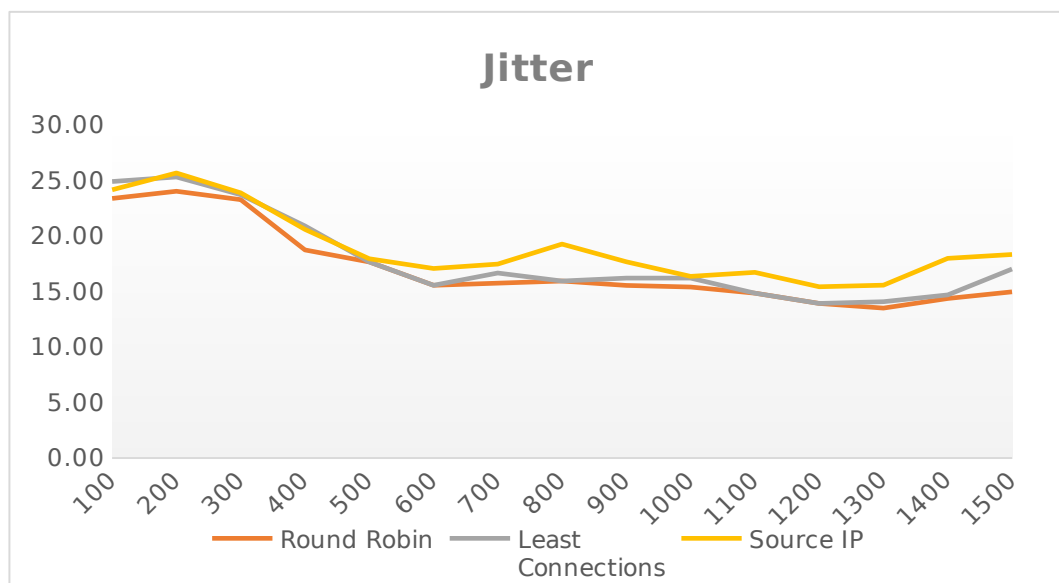
4.4.2.3 Pengujian Jitter

Pengujian parameter *jitter* memiliki kaitan erat dengan parameter *delay*. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan selisih antara *delay* pertama dengan *delay* selanjutnya, sehingga dari hasil pengamatan akan dilakukan perhitungan *jitter* dari setiap pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-2 dan dilakukan perhitungan *jitter* menggunakan persamaan (3.7). Hasil perhitungan *jitter* dapat dilihat pada tabel 4.13 berikut.

Tabel 4.13 Hasil Perhitungan *Jitter* Jaringan Sepi

<i>Request</i>	<i>Jitter (ms)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	23.40	24.93	24.19
200	24.04	25.33	25.69
300	23.29	23.74	23.90
400	18.76	20.93	20.61
500	17.68	17.68	17.96
600	15.57	15.57	17.09
700	15.76	16.68	17.48
800	15.95	15.95	19.28
900	15.56	16.22	17.69
1000	15.42	16.22	16.38
1100	14.87	14.87	16.74
1200	13.93	13.93	15.43
1300	13.52	14.09	15.59
1400	14.37	14.70	17.99
1500	14.99	17.04	18.35
Rata-rata	17.14	17.86	18.96

Berdasarkan Tabel 4.13 hasil perhitungan *jitter*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *jitter* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.23 grafik hasil pengujian *jitter* jaringan lokal berikut ini.

**Gambar 4.23** Hasil pengujian *jitter* jaringan sepi

Berdasarkan hasil pengujian parameter *jitter* jaringan sepi terhadap *web server*, nilai *jitter* tertinggi terdapat pada *web server source ip* yaitu sebesar 18.96 ms dan memiliki grafik *jitter* yang tidak stabil dan cenderung menurun. Nilai *jitter* terendah terdapat pada *web server round robin* dan memiliki perbedaan yang sangat kecil terhadap nilai *jitter* pada *web server least connections* yaitu sebesar 0,72 ms. Kedua *web server* ini memiliki trafik *delay* yang cenderung menurun ketika jumlah *request* ditambah pada setiap kali pengujian. Berdasarkan kategori degradasi *jitter* pada *load server* termasuk kategori bagus karena nilai *jitter* pada semua *web server* antara 0 hingga 75 ms.

4.4.2.4 Pengujian Packet Loss

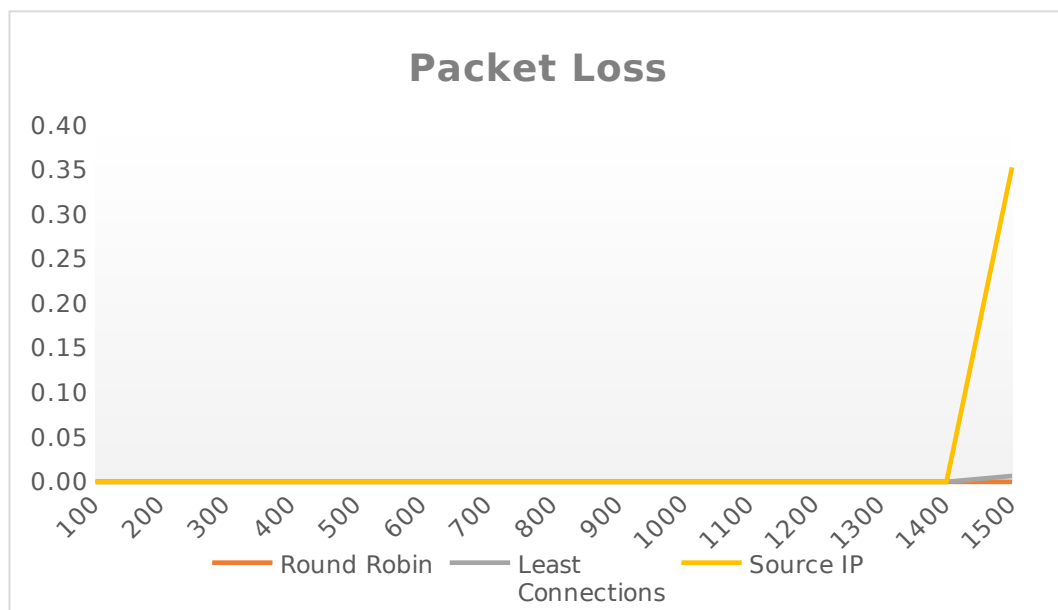
Pengujian parameter *packet loss* dilakukan untuk mengetahui jumlah *request* yang dapat ditangani oleh *web server*. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan *request* yang berhasil ditangani oleh *web server* setiap kali pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-1 dan dilakukan perhitungan nilai *packet loss* menggunakan persamaan (3.8). Hasil perhitungan *Packet loss* dapat dilihat pada tabel 4.14 berikut.

Tabel 4.14 Hasil Perhitungan *Packet Loss* Jaringan Sepi

<i>Request</i>	<i>Packet Loss (%)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	0.00	0.00	0.00
200	0.00	0.00	0.00
300	0.00	0.00	0.00
400	0.00	0.00	0.00
500	0.00	0.00	0.00
600	0.00	0.00	0.00
700	0.00	0.00	0.00
800	0.00	0.00	0.00
900	0.00	0.00	0.00
1000	0.00	0.00	0.00
1100	0.00	0.00	0.00
1200	0.00	0.00	0.00
1300	0.00	0.00	0.00
1400	0.00	0.00	0.00

Request	Packet Loss (%)		
	Round Robin	Least Connections	Source IP
1500	0.00	0.01	0.35
Rata-rata	0.00	0.00	0.02

Berdasarkan Tabel 4.14 hasil perhitungan *packet loss*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *packet loss* dari setiap server dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.24 grafik hasil pengujian *packet loss* jaringan lokal berikut ini.



Gambar 4.24 Hasil pengujian *packet loss* jaringan sepi

Berdasarkan hasil pengujian parameter *paket loss* terhadap web server pada jaringan sepi, nilai *packet loss* terndah terdapat pada web server round robin dan least connections yaitu sebesar 0% serta memiliki grafik yang stabil dan tidak terjadi kegagalan transaksi dari 100 hingga 1500 request. Nilai *packet loss* tertinggi terdapat pada web server source ip yaitu sebesar 0.35% dan grafik yang meningkat pada 1500 request. Nilai *packet loss* pada load server termasuk kategori sangat bagus, karena nilai *packet loss* berada pada 0 hingga 2% sesuai dengan kategori degradasi *packet loss*.

4.4.3 Hasil Pengujian *Quality of Service* Jaringan Sibuk

Pengujian *quality of service* dilakukan dengan cara mengamati kinerja *web server* pada saat diberikan beban dengan jumlah yang ditentukan. Pengujian dilakukan menggunakan jaringan sibuk dengan memberikan gangguan kepada *web server* untuk mendapatkan nilai dari parameter *throughput*, *delay*, *jitter* dan *packet loss*. Pengujian pada jaringan sibuk yaitu melakukan pengujian terhadap *web server* ketika jaringan unta2018 diakses oleh banyak pengguna pada jam 10:00 – 12:00 siang dan jaringan berada pada trafik yang tidak stabil. Hasil pengujian yang diperoleh melalui *tools siege*. Adapun hasil pengujian yang didapat sebagai berikut.

4.4.3.1 Pengujian *Throughput*

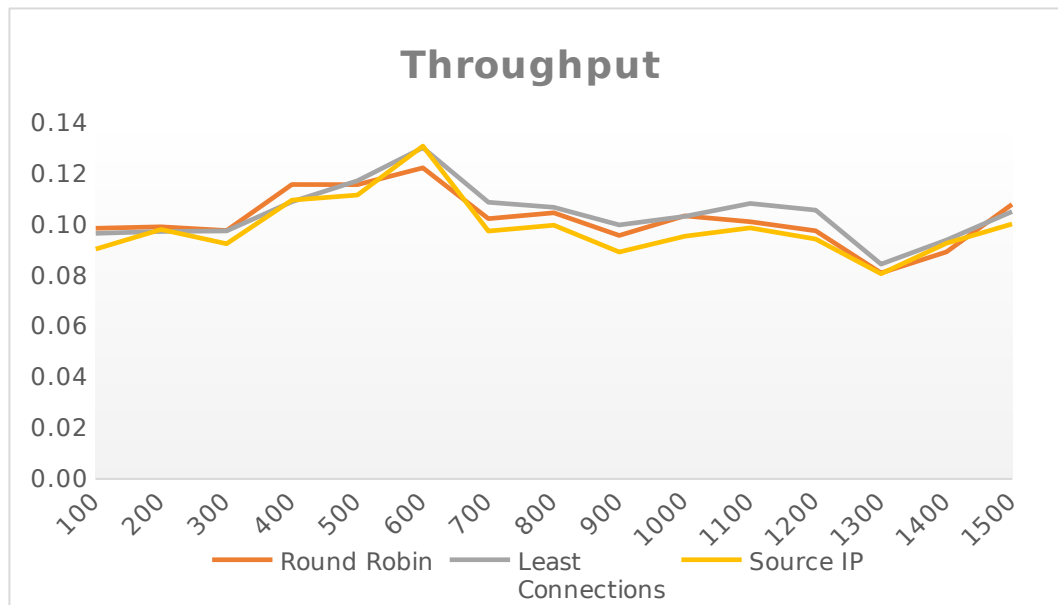
Pengujian parameter *throughput* dilakukan untuk mengetahui kemampuan *web server* dalam memberikan layanan secara benar terhadap *request* yang datang secara bersamaan. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan pada waktu yang diperlukan oleh *web server* dalam menangani *request* dan jumlah *bandwidth* yang diperlukan selama pengujian, sehingga dari hasil pengamatan akan dilakukan perhitungan *throughput* dari setiap kali pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-3 dan dilakukan perhitungan *throughput* menggunakan persamaan (3.4). Hasil perhitungan *throughput* dapat dilihat pada tabel 4.15 berikut.

Tabel 4.15 Hasil Perhitungan *Throughput* Jaringan Sibuk

<i>Request</i>	<i>Throughput (MBps)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	0.10	0.10	0.09
200	0.10	0.10	0.10
300	0.10	0.10	0.09
400	0.12	0.11	0.11
500	0.12	0.12	0.11
600	0.12	0.13	0.13
700	0.10	0.11	0.10
800	0.10	0.11	0.10
900	0.10	0.10	0.09
1000	0.10	0.10	0.10

Request	Throughput (Mbps)		
	Round Robin	Least Connections	Source IP
1100	0.10	0.11	0.10
1200	0.10	0.11	0.09
1300	0.08	0.08	0.08
1400	0.09	0.09	0.09
1500	0.11	0.11	0.10
Rata-rata	0.10	0.10	0.10

Berdasarkan Tabel 4.15 hasil perhitungan *throughput*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *throughput* dari setiap server dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.25 grafik hasil pengujian *throughput* jaringan sepi berikut ini.



Gambar 4.25 Hasil pengujian *throughput* jaringan sibuk

Berdasarkan hasil pengujian parameter *throughput* terhadap web server pada jaringan sibuk, web server round robin, least connections dan source ip memiliki nilai *throughput* sama yaitu sebesar 0,10 Mbps dan memiliki grafik yang meningkat pada 100 hingga 600 request dan grafik cenderung menurun pada request 700 hingga 1300, serta memiliki grafik yang meningkat pada request 1400 hingga 1500. Grafik *throughput* pada semua web server cenderung tidak stabil hal ini dikarenakan penggunaan jaringan pada kondisi sibuk tidak stabil.

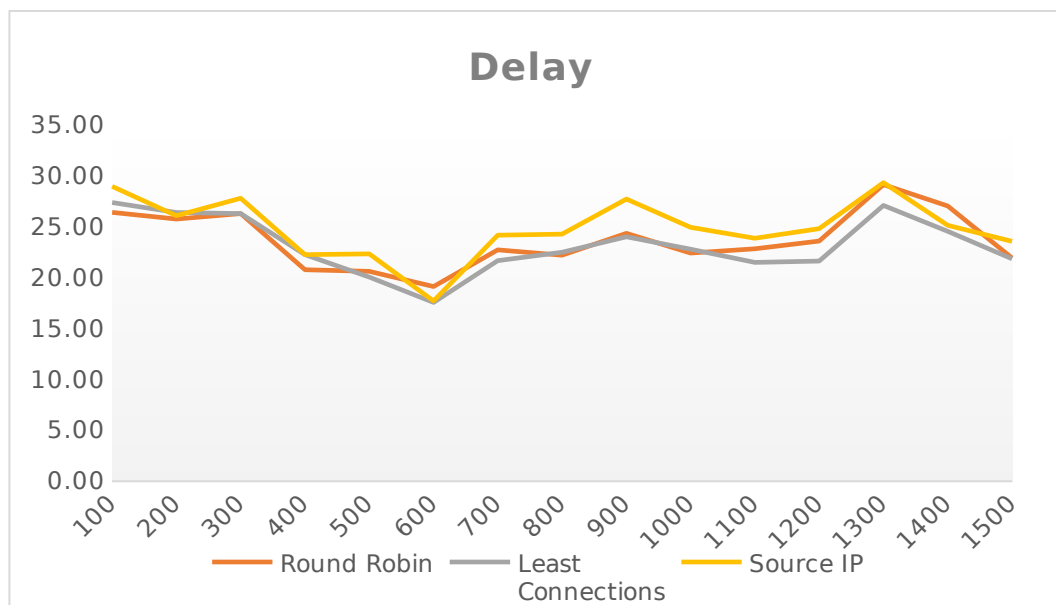
4.4.3.2 Pengujian Delay

Pengujian parameter *delay (legacy)* dilakukan untuk mengetahui waktu yang diperlukan untuk mengirim sejumlah *request* ke *web server*. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan jumlah waktu yang diperlukan selama pengiriman *request* dan jumlah *request* yang dapat diterima oleh *web server* selama pengujian, sehingga dari hasil pengamatan akan dilakukan perhitungan *delay* dari setiap pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-3 dan dilakukan perhitungan *delay* menggunakan persamaan (3.5). Hasil perhitungan *delay* dapat dilihat pada tabel 4.16 berikut.

Tabel 4.16 Hasil Perhitungan *Delay* Jaringan Sibuk

<i>Request</i>	<i>Delay (ms)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	26.43	27.41	29.00
200	25.78	26.42	26.11
300	26.32	26.32	27.82
400	20.79	22.27	22.27
500	20.64	20.08	22.36
600	19.14	17.58	17.73
700	22.74	21.69	24.20
800	22.23	22.51	24.29
900	24.38	24.03	27.75
1000	22.44	22.81	24.96
1100	22.85	21.51	23.88
1200	23.62	21.64	24.83
1300	29.16	27.12	29.36
1400	27.06	24.57	25.16
1500	21.95	21.87	23.59
Rata-rata	23.70	23.19	24.89

Berdasarkan Tabel 4.16 hasil perhitungan *delay*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *delay* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.26 grafik hasil pengujian *delay* jaringan lokal berikut ini.



Gambar 4.26 Hasil pengujian *delay* jaringan sibuk

Berdasarkan hasil pengujian parameter *delay* terhadap *web server* pada jaringan sibuk, nilai *delay* terendah terdapat pada *web server least connections* yaitu sebesar 23.19 ms dan memiliki selisih yang kecil terhadap nilai *delay* pada *web server round robin* yaitu sebesar 0,51 ms. Nilai *delay* pada *web server source ip* memiliki nilai yang terbesar diantara nilai *delay load server* yaitu sebesar 24,89 ms. Nilai *delay* pada *load server* memiliki grafik yang cenderung tidak stabil. Berdasarkan kategori latensi *delay*, nilai *delay load server* terdapat pada kategori sangat bagus, diakrenakan kurang dari 150 ms.

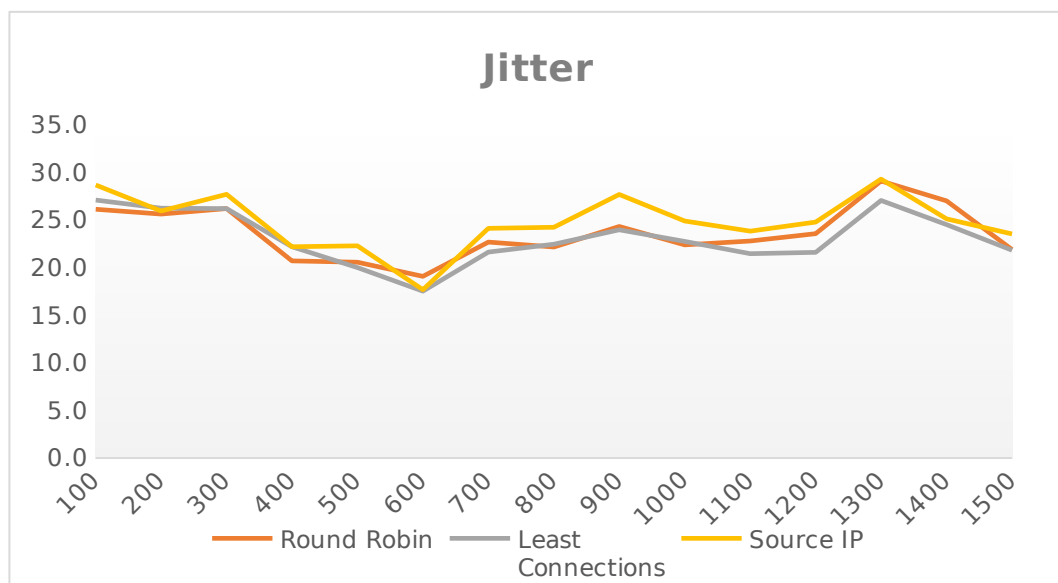
4.4.3.3 Pengujian *Jitter*

Pengujian parameter *jitter* memiliki kaitan erat dengan parameter *delay*. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan selisih antara *delay* pertama dengan *delay* selanjutnya, sehingga dari hasil pengamatan akan dilakukan perhitungan *jitter* dari setiap pengujian, sehingga dari hasil pengamatan akan dilakukan perhitungan *jitter* dari setiap pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-3 dan dilakukan perhitungan *jitter* menggunakan persamaan (3.7). Hasil perhitungan *jiiter* dapat dilihat pada tabel 4.17 berikut.

Tabel 4.17 Hasil Perhitungan *Jitter* Jaringan Sibuk

<i>Request</i>	<i>Jitter (ms)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	26.2	27.1	28.71
200	25.7	26.3	25.98
300	26.2	26.2	27.73
400	20.7	22.2	22.21
500	20.6	20.0	22.32
600	19.1	17.5	17.70
700	22.7	21.7	24.16
800	22.2	22.5	24.26
900	24.4	24.0	27.72
1000	22.4	22.8	24.94
1100	22.8	21.5	23.86
1200	23.6	21.6	24.81
1300	29.1	27.1	29.33
1400	27.0	24.5	25.15
1500	21.9	21.9	23.57
Rata-rata	23.65	23.13	24.83

Berdasarkan Tabel 4.17 hasil perhitungan *jitter*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *jitter* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.27 grafik hasil pengujian *jitter* jaringan lokal berikut ini.

**Gambar 4.27** Hasil pengujian *jitter* jaringan sibuk

Berdasarkan hasil pengujian parameter *jitter* jaringan sibuk terhadap *web server*, nilai *jitter* terendah terdapat pada *server least connections* yaitu sebesar 23,13 ms dan memiliki selisih yang kecil terhadap *web server round robin* yaitu sebesar 0,52 ms. Nilai *jitter* tertinggi terdapat pada *web server source ip* yaitu sebesar 24,83 ms. Nilai *jitter* pada *load server* memiliki grafik yang tidak stabil. Berdasarkan kategori degradasi *jitter*, nilai *jitter load server* terdapat pada kategori bagus karena berada pada 0 hingga 75 ms, berdasarkan kategori degradasi *jitter*.

4.4.3.4 Pengujian *Packet Loss*

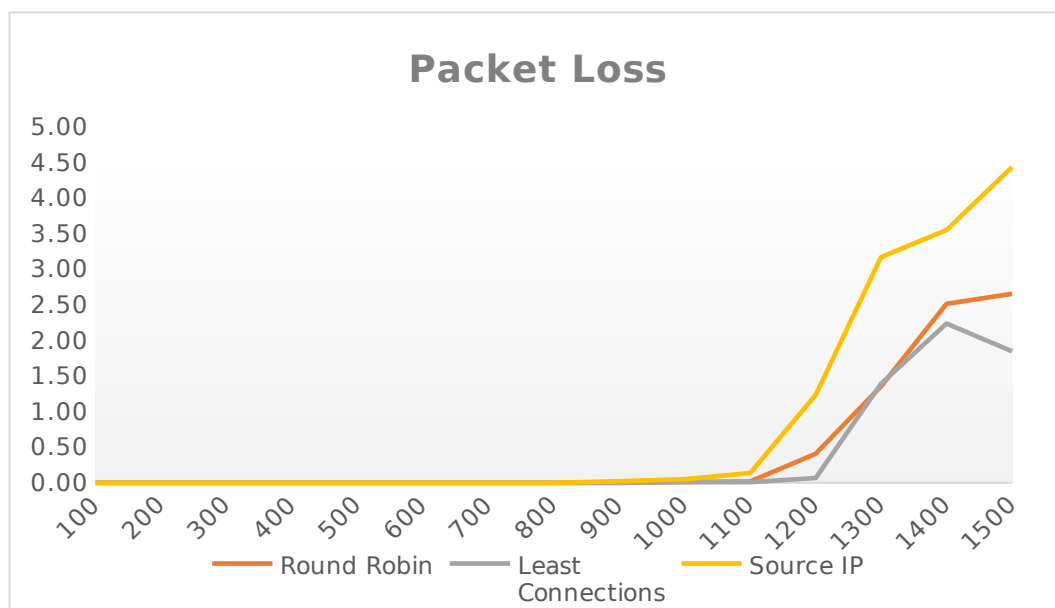
Pengujian parameter *packet loss* dilakukan untuk mengetahui jumlah *request* yang dapat ditangani oleh *web server*. Pengujian ini dilakukan untuk mendapatkan hasil pengamatan *request* yang berhasil ditangani oleh *web server* setiap kali pengujian. Data hasil pengamatan dapat dilihat pada lampiran B-3 dan dilakukan perhitungan nilai *packet loss* menggunakan persamaan (3.8). Hasil perhitungan *Packet loss* dapat dilihat pada tabel 4.18 berikut.

Tabel 4.18 Hasil Perhitungan *Packet Loss* Jaringan Sibuk

<i>Request</i>	<i>Packet Loss (%)</i>		
	<i>Round Robin</i>	<i>Least Connections</i>	<i>Source IP</i>
100	0.00	0.00	0.00
200	0.00	0.00	0.00
300	0.00	0.00	0.00
400	0.00	0.00	0.00
500	0.00	0.00	0.00
600	0.00	0.00	0.00
700	0.00	0.00	0.00
800	0.00	0.00	0.00
900	0.00	0.00	0.02
1000	0.01	0.01	0.05
1100	0.02	0.01	0.14
1200	0.41	0.07	1.23
1300	1.35	1.39	3.17
1400	2.51	2.24	3.55
1500	2.65	1.85	4.43

Request	Packet Loss (%)		
	Round Robin	Least Connections	Source IP
Rata-rata	0.46	0.37	0.84

Berdasarkan Tabel 4.18 hasil perhitungan *packet loss*, dibuat grafik untuk menampilkan perbandingan hasil pengujian *packet loss* dari setiap server dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.28 grafik hasil pengujian *packet loss* jaringan lokal berikut ini.



Gambar 4.28 Hasil pengujian *packet loss* jaringan sibuk

Berdasarkan hasil pengujian parameter *paket loss* terhadap web server pada jaringan sibuk, nilai *packet loss* terendah terdapat pada web server *least connections* yaitu sebesar 0,37%. Nilai *packet loss* pada web server *round robin* berada lebih tinggi dari nilai *packet loss round robin* dan berada dibawah *source ip* yaitu sebesar 0,46%. Nilai *packet loss* tertinggi terdapat pada web server *source ip* yaitu sebesar 0,84%. Grafik *packet loss* pada *load server* memiliki peningkatan pada *request* 1100 hingga 1500 *request*. Berdasarkan kategori degradasi pada *packet loss*, nilai *packet loss load server* termasuk kategori sangat bagus.

4.4.4 Analisis Hasil Pengujian *Quality of Service*

Berdasarkan hasil pengujian *quality of service* dari parameter *throughput*, *delay*, *jitter* dan *packet loss* yang dilakukan pengujian antara *load server* pada tiga jenis jaringan yaitu jaringan lokal, jaringan sepi dan jaringan sibuk didapatkan hasil yang masing-masing dari setiap jenis jaringan berbeda-beda karena sesuai dengan kondisi jaringan dan kemampuan *web server* dalam menangani jumlah *request* yang dikirim dari 100 hingga 1500 *request*.

Berdasarkan hasil pengujian *quality of service* pada jaringan lokal dari empat parameter yang dilakukan pengujian, pada jaringan lokal yang tidak memiliki koneksi internet *web server* dengan metode *round robin* bekerja secara maksimal sehingga menghasilkan nilai parameter *throughput* tertinggi yaitu sebesar 0,14 Mbps, nilai *delay* terendah yaitu sebesar 17,21 ms dan nilai *jitter* terendah yaitu sebesar 17,16 ms, serta nilai persentase *packet loss* terendah yaitu sebesar 0% dibandingkan dengan jenis *web server* pada metode *load balancer* lain.

Berdasarkan hasil pengujian *quality of service* jaringan unta2018 pada kondisi sepi dari empat parameter yang dilakukan pengujian, *web server* dengan metode *round robin* bekerja secara maksimal sehingga menghasilkan nilai parameter *throughput* tertinggi yaitu sebesar 0,14 Mbps sama dengan *least connections*, nilai *delay* terendah yaitu sebesar 17,19 ms dan nilai *jitter* terendah yaitu sebesar 17,14 ms, serta nilai persentase *packet loss* terendah dibandingkan dengan jenis *web server* pada metode *load balancer* lain yaitu sebesar 0%.

Berdasarkan hasil pengujian *quality of service* jaringan unta2018 pada kondisi sibuk dari empat parameter yang dilakukan pengujian, *web server* dengan metode *least connections* menghasilkan nilai parameter *throughput* sama dengan *web server* lain yaitu sebesar 0,10 Mbps, nilai *delay* terendah yaitu sebesar 23,19 ms dan nilai *jitter* terendah yaitu sebesar 23,13 ms, serta nilai persentase *packet loss* terendah dibandingkan dengan jenis *web server* lain yaitu sebesar 0,37%.

Pengujian *quality of service* pada jaringan lokal dan jaringan sepi, *web server* yang menggunakan algoritma *round robin* lebih unggul dari *web server* lain. Metode *round robin* bekerja dengan cara membagi jumlah *request* secara merata kepada *web server* yang tersedia, sehingga ketika jaringan stabil algoritma ini

bekerja secara maksimal dari metode *least connections* dan *source ip*. Metode *round robin* sangat berpengaruh terhadap kestabilan jaringan dan kondisi *web server* dalam keadaan tidak terjadi *down*. *Web server* pada jaringan sibuk menggunakan algoritma *least connections* lebih unggul dari algoritma lain, hal ini dikarenakan karakter dari metode *least connections* membagi *request* yang dikirim kepada *web server* dengan cara memprioritaskan *web server* yang memiliki koneksi paling sedikit dan tidak berpengaruh terhadap kondisi jaringan yang tidak stabil dan kondisi *web server* terjadi *down*.

4.5 Hasil Pengujian Workload

Pengujian *workload* dilakukan untuk mengetahui batas kemampuan *web server* dapat menangani jumlah *request* yang dikirim pada saat dilakukan pengujian. Pengujian ini dilakukan dengan cara mengirim *request* mulai dari 100, 200, hingga *request* yang dikirim tidak dapat ditangani oleh *web server* dan terjadi *fail transaction*. Setiap *request* yang dikirim diulang sebanyak sepuluh kali. Pengujian ini dilakukan pada tiga kondisi jaringan yaitu kondisi jaringan lokal, jaringan sepi dan jaringan sibuk. Tujuan pengujian *workload* untuk mengetahui kemampuan dari setiap *web server* dalam menangani *request* yang dikirim. Adapun pengujian dilakukan sebagai berikut.

4.5.1 Hasil Pengujian Workload Jaringan Lokal

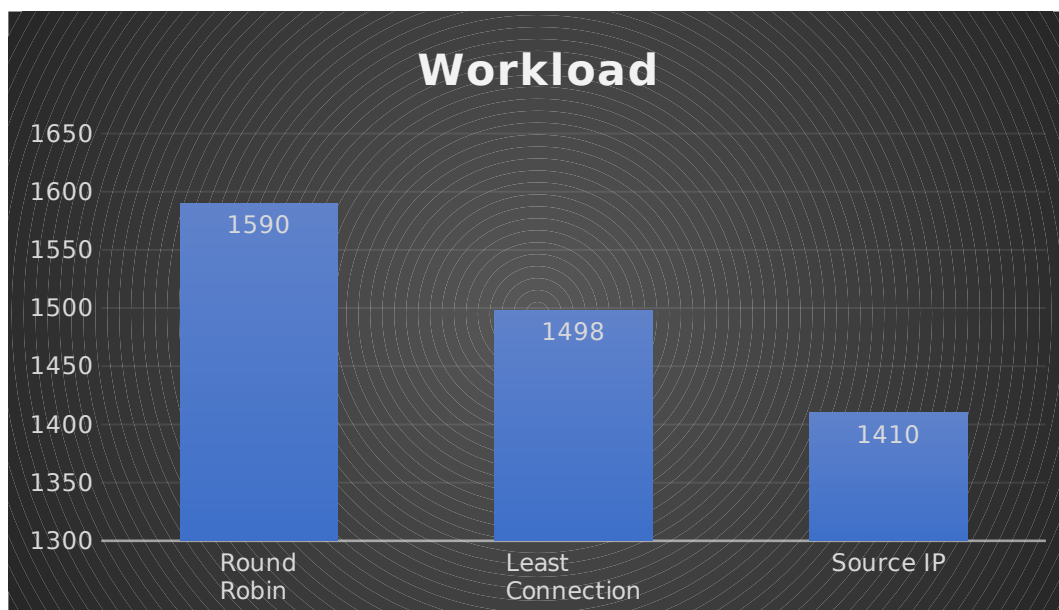
Pengujian *workload* dilakukan dengan cara mengamati kemampuan *web server* dalam menangani *request* yang dikirim hingga terjadi *fail transaction* saat dilakukan pengujian. Pengujian dilakukan menggunakan jaringan lokal dengan memberikan gangguan kepada *web server* untuk mendapatkan *request* maksimal yang dapat ditangani oleh *web server*. Hasil pengujian yang diperoleh melalui *tools siege* terdapat pada lampiran C-1 ditampilkan pada tabel 4.19 berikut.

Tabel 4.19 Hasil Pengujian Workload Jaringan Lokal

Request	Round Robin		Least Connections		Source IP	
	Successful Transactions	Failed Transactions	Successful Transactions	Failed Transactions	Successful Transactions	Failed Transactions
100	100	0	100	0	100	0
200	200	0	200	0	200	0

Request	Round Robin		Least Connections		Source IP	
	Successful Transactions	Failed Transactions	Successful Transactions	Failed Transactions	Successful Transactions	Failed Transactions
300	300	0	300	0	300	0
400	400	0	400	0	400	0
500	500	0	500	0	500	0
600	600	0	600	0	600	0
700	700	0	700	0	700	0
798	798	0	798	0	798	0
800	800	0	800	0	800	0
900	900	0	900	0	900	0
1000	1000	0	1000	0	1000	0
1100	1100	0	1100	0	1100	0
1200	1200	0	1200	0	1200	0
1300	1300	0	1300	0	1300	0
1400	1400	0	1400	0	1400	0
1410	1410	0	1410	0	1410	0
1412	1412	0	1412	0	1384	28
1498	1498	0	1498	0	-	-
1500	1500	0	1498	2	-	-
1590	1590	0	-	-	-	-
1592	1591	1	-	-	-	-

Berdasarkan Tabel 4.19 hasil pengujian *workload* jaringan lokal, dibuat grafik untuk menampilkan perbandingan hasil pengujian *workload* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.29 grafik hasil pengujian *workload* jaringan lokal berikut ini.



Gambar 4.29 Hasil pengujian *workload* jaringan lokal

Berdasarkan hasil pengujian *workload* jaringan lokal, jumlah *request* tertinggi yang dapat ditangani oleh *web server* dengan algoritma *round robin* yaitu sebesar 1590 *request*, sedangkan pada *web server* dengan algoritma *least connections* sebesar 1498 *request* dan algoritma *source ip* memiliki jumlah *request* terendah yaitu sebesar 1410 *request*.

4.5.2 Hasil Pengujian *Workload* Jaringan Sepi

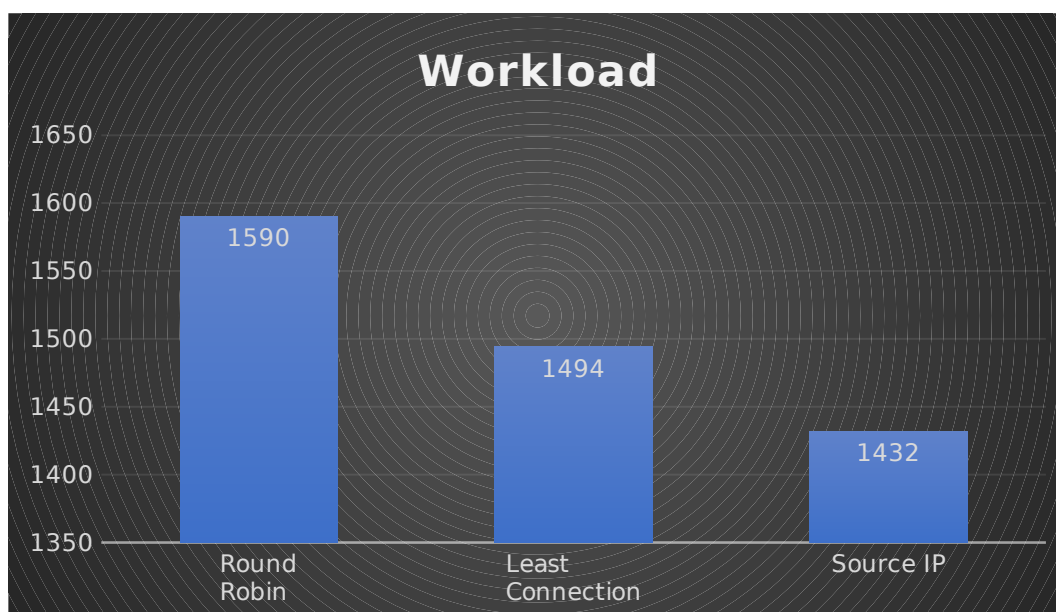
Pengujian *workload* dilakukan dengan cara mengamati kemampuan *web server* dalam menangani *request* yang dikirim hingga terjadi *fail transaction* saat dilakukan pengujian. Pengujian dilakukan menggunakan jaringan sepi dengan memberikan gangguan kepada *web server* untuk mendapatkan *request* maksimal yang dapat ditangani oleh *web server*. Hasil pengujian yang diperoleh melalui *tools siege* terdapat pada lampiran C-2 ditampilkan pada tabel 4.20 berikut.

Tabel 4.20 Hasil Pengujian *Workload* Jaringan Sepi

<i>Request</i>	<i>Round Robin</i>		<i>Least Connections</i>		<i>Source IP</i>	
	<i>Successful Transactions</i>	<i>Failed Transactions</i>	<i>Successful Transactions</i>	<i>Failed Transactions</i>	<i>Successful Transactions</i>	<i>Failed Transactions</i>
100	100	0	100	0	100	0
200	200	0	200	0	200	0
300	300	0	300	0	300	0
400	400	0	400	0	400	0
500	500	0	500	0	500	0
600	600	0	600	0	600	0
700	700	0	700	0	700	0
800	800	0	800	0	800	0
898	898	0	898	0	898	0
900	900	0	900	0	900	0
1000	1000	0	1000	0	1000	0
1100	1100	0	1100	0	1100	0
1200	1200	0	1200	0	1200	0
1300	1300	0	1300	0	1300	0
1400	1400	0	1400	0	1400	0
1432	1432	0	1432	0	1432	0
1434	1434	0	1434	0	1409	25
1494	1494	0	1494	0	-	-

Request	Round Robin		Least Connections		Source IP	
	Successful Transactions	Failed Transactions	Successful Transactions	Failed Transactions	Successful Transactions	Failed Transactions
1496	1496	0	1436	60	-	-
1590	1590	0	-	-	-	-
1592	1591	1	-	-	-	-

Berdasarkan Tabel 4.20 hasil pengujian *workload* jaringan sepi, dibuat grafik untuk menampilkan perbandingan hasil pengujian *workload* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.30 grafik hasil pengujian *workload* jaringan sepi berikut ini.



Gambar 4.30 Hasil pengujian *workload* jaringan sepi

Berdasarkan hasil pengujian *workload* jaringan sepi, jumlah *request* tertinggi yang berhasil ditangani oleh *web server* yaitu terdapat pada *web server* menggunakan algoritma *round robin* yaitu sebesar 1590 *request*. *Web server* dengan algoritma *least connections* berada dibawah metode *round robin* dengan jumlah *request* yang berhasil ditangani sebesar 1494 *request*. *Request* terendah yang dapat ditangani oleh *web server* terdapat pada algoritma *source ip* yaitu sebesar 1432 *request*.

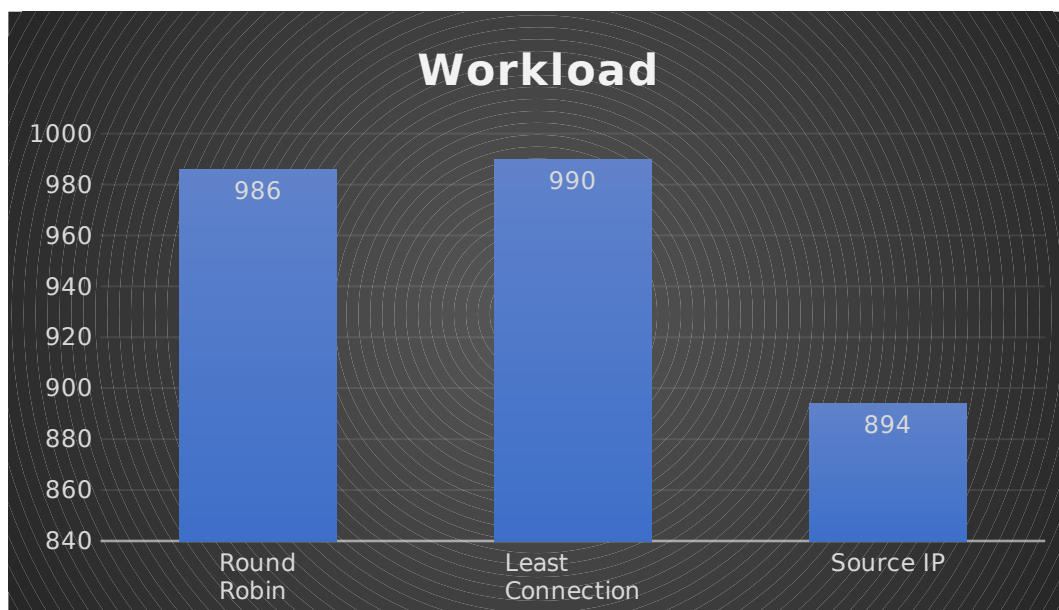
4.5.3 Hasil Pengujian *Workload* Jaringan Sibuk

Pengujian *workload* dilakukan dengan cara mengamati kemampuan *web server* dalam menangani *request* yang dikirim hingga terjadi *fail transaction* saat dilakukan pengujian. Pengujian dilakukan menggunakan jaringan sibuk dengan memberikan gangguan kepada *web server* untuk mendapatkan *request* maksimal yang dapat ditangani oleh *web server*. Hasil pengujian yang diperoleh melalui *tools siege* terdapat pada lampiran C-3 ditampilkan pada tabel 4.21 berikut.

Tabel 4.21 Hasil Pengujian *Workload* Jaringan Sibuk

Request	Round Robin		Least Connections		Source IP	
	Successful Transactions	Failed Transactions	Successful Transactions	Failed Transactions	Successful Transactions	Failed Transactions
100	100	0	100	0	100	0
200	200	0	200	0	200	0
300	300	0	300	0	300	0
400	400	0	400	0	400	0
500	500	0	500	0	500	0
598	598	0	598	0	598	0
600	600	0	600	0	600	0
700	700	0	700	0	700	0
800	800	0	800	0	800	0
892	892	0	892	0	892	0
894	894	0	894	0	893	1
900	900	0	900	0	-	-
986	986	0	986	0	-	-
988	986	2	988	0	-	-
990	-	-	990	0	-	-
992	-	-	989	3	-	-

Berdasarkan Tabel 4.21 hasil pengujian *workload* jaringan sibuk, dibuat grafik untuk menampilkan perbandingan hasil pengujian *workload* dari setiap *server* dalam bentuk visual. Adapun grafik yang dimaksud dapat dilihat pada Gambar 4.31 grafik hasil pengujian *workload* jaringan sibuk berikut ini.

**Gambar 4.31** Hasil pengujian *workload* jaringan sibuk

Berdasarkan hasil pengujian *workload* pada jaringan sibuk, jumlah *request* tertinggi yang dapat ditangani oleh *web server* dengan algoritma *least connections* yaitu sebesar 990 *request* dan memiliki perbedaan 4 *request* lebih tinggi terhadap *web server* dengan algoritma *round robin*. *Web server* dengan algoritma *source ip* memiliki jumlah *request* yang paling rendah terhadap *web server* lain yaitu sebesar 892 *request*.

4.5.4 Analisis Hasil Pengujian *Workload*

Berdasarkan hasil pengujian pada kondisi jaringan lokal dan jaringan sepi, *web server* yang menggunakan algoritma *round robin* memiliki jumlah *request* yang dapat ditangani oleh *web server* lebih tinggi dibanding dengan *web server* lain yaitu sebesar 1590 *request*. Hal ini dikarenakan *web server* yang menggunakan algoritma *round robin* bekerja secara optimal pada kondisi jaringan yang stabil dan kondisi *web server* dalam keadaan tidak terjadi *down* sehingga *request* yang dikirim dalam jumlah besar dapat ditangani dengan maksimal dan memperkecil terjadinya kegagalan transaksi pada *web server*.

Web server yang menggunakan algoritma *least connections* lebih unggul pada kondisi jaringan sibuk memiliki *request* yang terbanyak yang dapat ditangani oleh *web server* yaitu sebesar 990 *request*, hal ini dikarenakan kemampuan *web server* yang menggunakan algoritma *least connectons* bekerja secara optimal pada kondisi jaringan yang tidak stabil, sehingga memperkecil terjadinya *request* yang gagal ditangnai oleh *web server*.

4.6 Analisis Keseluruhan

Berdasarkan hasil pengujian yang telah dilakukan menunjukkan bahwa sistem *load balancer* yang diterapkan pada *instance* openstack bekerja secara optimal dan sesuai dengan karakteristik dari metode *load balancing*. Pengujian *availability* bertujuan untuk mengetahui kemampuan *web server* ketika diberikan gangguan berupa pengiriman *request* dalam jumlah besar terhadap *web server* pada *instance* yang menggunakan metode *load balancing*. Hasil pengujian *availability* menunjukkan *web server* yang menggunakan algoritma *round robin* lebih unggul dalam skenario pertama pada kondisi jaringan lokal dan normal yaitu

sebesar 99,76% dan 98,21%. Hal ini dikarenakan algortima *round robin* membagi beban yang dikirim secara merata terhadap *web server* yang tersedia dan bekerja secara maksimal pada jaringan yang stabil dan memiliki beban *request* yang tidak begitu besar. Nilai *availability* tertinggi pada skenario kedua dan ketiga pada jaringan lokal yaitu terdapat pada *web server least connections* sebesar 97,63% dan 94,73%. Nilai *availability* tertinggi pada skenario kedua dan ketiga pada jaringan normal yaitu terdapat pada *web server least connections* sebesar 95,27% dan 93,19%. Nilai *availability* tertinggi pada tiga skenario dan dalam jaringan sibuk terdapat pada *web server least connections* yaitu 91,85% skenario pertama, 91,09% skenario kedua dan 90,65% skenario ketiga. Algortima *least connections* bekerja dengan maksimal ketika *web server* terjadi *down*, jaringan tidak stabil dan memiliki beban *request* yang besar.

Berdasarkan hasil pengujian *quality of service* yang bertujuan untuk menentukan kualitas dari layanan jaringan menggunakan parameter *throughput*, *delay*, *jitter* dan *packet loss*. Hasil pengujian *quality of service* pada jaringan lokal, nilai *throughput* tertinggi yaitu sebesar 0,14 Mbps terdapat pada *web server round robin* dan *least connections*, nilai *delay* terendah yaitu sebesar 17,21 ms dan nilai *jitter* terendah yaitu sebesar 17,16 ms terdapat pada *web server round robin* dan nilai *packet loss* terendah terdapat pada *web server round robin* dan *least connections* yaitu sebesar 0%. Hasil pengujian *quality of service* pada jaringan sepi, nilai *throughput* tertinggi yaitu sebesar 0,14 Mbps terdapat pada *web server round robin* dan *least connections*, nilai *delay* terendah yaitu sebesar 17,19 ms dan nilai *jitter* terendah yaitu sebesar 17,14 ms terdapat pada *web server round robin* dan nilai *packet loss* terendah terdapat pada *web server round robin* dan *least connections* yaitu sebesar 0%. Hasil pengujian *quality of service* pada jaringan sibuk, nilai *throughput laod server* sama yaitu sebesar 0,10 Mbps, nilai *delay* terendah yaitu sebesar 23,19 ms dan nilai *jitter* terendah yaitu sebesar 23,13 ms terdapat pada *web server least connections* dan nilai *packet loss* terendah terdapat pada *web server least connections* yaitu sebesar 0,37%.

Pengujian *workload* bertujuan untuk menentukan batas maksimal *web server* dalam menangani beban yang dikirim dalam jumlah besar agar tidak terjadi kegagalan *request* yang dikirim oleh *client*. Hasil pengujian *workload* pada

jaringan lokal dan sepi, *web server round robin* memiliki nilai *workload* tertinggi yaitu sebesar 1590 *request* sedangkan pada *web server least connections* memiliki nilai *workload* tertinggi yaitu sebesar 990 yang terdapat pada jaringan sibuk.

Berdasarkan nilai dari setiap parameter pada pengujian *quality of service* dan *workload*, maka *web server round robin* lebih unggul pada jaringan lokal dan sepi, hal ini dikarenakan algoritma *round robin* bekerja secara maksimal pada kondisi jaringan yang stabil dan memiliki trafik yang rendah. Sedangkan pada *web server least connections* bekerja secara maksimal pada kondisi jaringan sibuk yang memiliki trafik yang tinggi dan kondisi jaringan tidak stabil sehingga mampu memberikan layanan yang maksimal terhadap *client*.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil pengujian dan analisa terhadap performa pada *web server* yang menggunakan algoritma *round robin*, *least connections* dan *source ip* pada kondisi jaringan lokal, jaringan sepi dan jaringan sibuk, maka dapat diambil kesimpulan sabagai berikut.

1. Hasil pengujian *availability* pada *web server* yang menggunakan algoritma *least connections* lebih unggul yaitu dengan rata-rata nilai *availability* sebesar 94,67 %, hal ini dikarenakan metode ini bekerja dengan baik saat *web server* terjadi *down* dan jaringan tidak stabil, serta mampu menangani *request* dalam jumlah besar, sehingga nilai *availability* lebih tinggi dari *web server* lain.
2. Hasil pengujian *quality of service* pada *web server* yang menggunakan algoritma *round robin* lebih unggul pada kondisi jaringan stabil dan *web server* dalam kondisi tidak terjadi *down*. Sedangkan pada kondisi jaringan tidak stabil, hasil pengujian *quality of service* pada *web server* yang menggunakan metode *least connections* lebih unggul dari *round robin* atau *web server* yang lain.
3. Hasil pengujian *workload* pada *web server* yang menggunakan algoritma *round robin* mampu menangani beban *request* yaitu sebesar 1590 *request* pada jaringan lokal dan sepi, sedangkan algoritma *least connections* mampu menangani beban *request* yaitu sebesar 990 *request* pada jaringan sibuk.
4. Berdasarkan hasil pengujian yang telah dilakukan pada perancangan *cloud computing* yang berbasis infrastruktur dan menggunakan metode *load balancing*, direkomendasikan untuk menggunakan algoritma *round robin* pada *web server* apabila kondisi jaringan relatif stabil dan *web server* tidak sering terjadi *down* dan direkomendasikan menggunakan algoritma *least*

connections pada *web server* apabila kondisi jaringan relatif tidak stabil dan *web server* sering terjadi *down*.

5.2 Saran

Saran yang dapat diberikan untuk pengembangan penelitian analisa performa *load server* pada openstack adalah sebagai berikut.

1. Pada pengembangan penelitian selanjutnya hendaknya membuat suatu metode untuk menggabungkan algoritma *round robin* dan *least connections* yang terdapat pada *load balancer*, sehingga *web server* bisa menggunakan metode gabungan secara otomatis dan *web server* bisa bekerja lebih optimal.
2. Pada pengembangan penelitian selanjutnya agar menambah fitur docker dan menerapkan sistem kubernetes agar manajemen *web server* lebih praktis dan efisien serta memiliki kemampuan *load service* lebih baik.

DAFTAR PUSTAKA

- Adenan, R., Abdurohman, M., & Jadied, E. M. 2018. Analisis Perbandingan Algoritma Load Balancing Round Robin dan Least Connections pada Software Defined Network. *Telkom University Open Library*.
- Adrika, M. A., Perdana, D., & Sanjoyo, D. D. 2018. Perancangan dan Implementasi High-Availaility Layanann VoIP dengan Metoda Load Balancing as a Service pada Openstack Cloud. *Telkom University Open Library*.
- Anggeriana, H. 2011. *Cloud Computing*. DKI Jakarta: ITLinks indonesia.
- Anggeriana, H. 2011. Pengembangan Elemen Cloud Computing dalam Sistem Teknologi Informasi. *Journal of Information System & Technology*, ISSN 2085-8299.
- Ashari, A., & Setiawan, H. 2011. Cloud Computing : Solusi ICT ? *Jurnal Sistem Informasi (JSI)*, Vol. 3, No. 2, 336-345.
- Cakrawerdya, H. K., Mayasari, R., & Sanjoyo, D. D. 2017. Implementation load balancer as a service in openstack based on NFV. *J. Computer Applications in Technology*, Vol. XX, No. XX, XXXX, 240-245.
- Fajrin, T. 2012. Analisis Sistem Penyimpanan Data Menggunakan Sistem Cloud Computing Studi Kasus SMK N 2 Karanganyar. *IJNS*, Vol. 1, No. 1, 31-35.
- Iskandar, I., & Hidayat, A. 2015. Analisa Quality of Service (QoS) Jaringan Internet Kampus (Studi Kasus: UIN Suska Riau). *Jurnal CoreIT*, Vol. 1, No. 2, 67-76.
- Novianti, T., & Widianoro, A. 2016. Analisa QOS (Quality of Services) pada Implementasi IPV4 dan IPV6 dengan Teknik Tunneling. *Jurnal Imliah Rekayasa*, Vol. 9, No. 2, 76-83.
- Nugraha, P. 2016. Rancang Bangun Web Server Berbasis Linux Dengan Metode Load Balancing. *JUSTIN*, Vol. 4, No. 3, 1-5.
- Pratama, R. 2019. *Rancang Bangun Jaringan Eduroam di Universitas Tanjungpura*. Pontianak.
- Pribadi, Y. 2019. *Analisis Penggunaan Metode Failover Clustering Untuk Mencapai High Availability pada Web Server (Studi Kasus : Gedung Jurusan Informatika)*. Pontianak.
- Ramadhan, G., Latuconsina, R., & Purboyo, W. T. 2019. Analisis Performansi Load Blaancing pada Cloud Computing Menggunakan Algoritma Throttled dan Greedy. *Telkom University Open Library*.
- Sasmita, W. P., Safriadi, N., & Irwansyah, M. A. 2013. Analisis Quality of Service (QoS) Pada Jaringan Internet (Studi Kasus : Fakultas Kedokteran Universitas Tanjungpura). *Justin*, Vol. 1, No. 1, 1-6.

- Triyanto, H. 2019. *Analisis Perbandingan Performa Openstack dan Apache Cloudstack Dalam Model Cloud Computing Berbasis Infrastructure as a Service*. Pontianak.
- Umam, C., Handoko, L. B., & Rizqi, G. M. 2018. Implementation And Analysis High Availability Network File. *TRANSFORMATIKA*, Vol. 16, No. 1, 31-39.
- Yanto. 2013. Analisis QoS (Quality of Service) Pada Jaringan Internet (Studi Kasus Fakultas Teknik Untan). *JUSTIN*, Vol. 1, No. 1.
- Yuniati, Y., Fitriawan, H., & Patih, D. F. 2014. Analisis Perancangan Server VOIP (voice Over Internet Protocol) Dengan Opensource Asterik dan VPN (Virtual Private Network) Sebagai Pengaman Jaringan Antar Client. *Jurnal Sains, Teknologi dan Industri*, Vol.12, No. 1, 112-121.