Perbandingan Kinerja *HAproxy* Dan *Zevenet* Dalam Pengimplementasian *Multi Service Load Balancing*

Skripsi

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Faris Muslim Azmi
115060807111150

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

PENGESAHAN

PERBANDINGAN KINERJA HAPROXY DAN ZEVENET DALAM PENGIMPLEMENTASIAN MULTI SERVICE LOAD BALANCING

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Faris Muslim Azmi
NIM: 115060807111150

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

Dosen Pembimbing I	Dosen Pembimbing II
Mahendra Data, S.Kom., M.Kom	Heru Nurwarsito, Ir., M.Kom
NIK: 2015038611171001	NIP: 196504021990021001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 197105182003121002

IDENTITAS TIM PENGUJI

Majelis Penguji Ujian Skripsi

Ketua Majelis (Penguji I) : Widhi Yahya, S.Kom., M.Sc.
(NIK. 2016078911211001)

Penguji II : Fariz Andri Bakhtiar, S.T., M.Kom.
(NIK. 2017098403141001)

Pembimbing

Pembimbing I : Mahendra Data, S.Kom., M.Kom
(NIP: 2015038611171001)

Pembimbing II : Heru Nurwarsito, Ir., M.Kom
(NIP: 196504021990021001)

PERNYATAAN ORISINALITAS

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

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

Malang, 03 Agustus 2018

Faris Muslim Azmi
NIM: 115060807111150

DAFTAR RIWAYAT HIDUP

DATA PRIBADI

Nama Lengkap	: Faris Muslim Azmi
Tempat/Tanggal Lahir	: Denpasar, 14 Ferbruari 1993
Alamat	: Ds. Dalam RT.01/RW. 02, Kec. Alas, Sumbawa-NTB
Jenis Kelamin	: Laki-laki
Agama	: Islam
Kewarganegaraan	: Indonesia
Status	: Belum Menikah
No. HP	: 082340302121
Email	: bigfaris0@gmail.com

PENDIDIKAN FORMAL

1999-2000	TK Dharma Bakti Alas
2000-2007	SDN 07 Alas
2007-2009	SMPN 1 Alas
2009-2011	SMAN 1 Alas
2011-2018	Universitas Brawijaya Malang

Demikian daftar riwayat hidup ini penulis buat dengan sebenar-benarnya.

Malang, 05 Agustus 2018
Hormat Penulis

Faris Muslim Azmi

KATA PENGANTAR

Puji dan syukur Penulis panjatkan kehadirat Allah Subhanahu wa Ta'ala, karena hanya dengan rahmat dan karunia-Nya Penulis dapat menyelesaikan skripsi dengan judul **"Perbandingan Kinerja *HAproxy* Dan *Zevenet* Dalam Pengimplementasian *Multi Service Load Balancing*"** dengan baik. Melalui kesempatan ini, Penulis ingin menyampaikan rasa hormat dan terima kasih yang sebesar-besarnya kepada semua pihak yang telah memberikan bantuan dan dukungan selama pengerjaan skripsi, diantaranya:

1. Mahendra Data, S.Kom., M.Kom, selaku dosen pembimbing I yang telah banyak memberikan ilmu, bimbingan, arahan, motivasi, serta meluangkan waktunya selama penyusunan skripsi ini.
2. Heru Nurwarsito, Ir., M.Kom., selaku dosen pembimbing II yang telah banyak memberikan ilmu, bimbingan, arahan, nasihat, serta meluangkan waktunya selama penyusunan skripsi ini.
3. Tri Astoto Kurniawan, S.T, M.T, Ph.D., selaku Ketua Jurusan Teknik Informatika Universitas Brawijaya.
4. Agus Wahyu Widodo, S.T, M.Cs., Ketua Program Studi Teknik Informatika Universitas Brawijaya.
5. Kedua orang tua Ir. Faesal dan dra. Masnawati yang telah memberi motivasi, kasih sayang, doa serta dukungan moril dan materil. Kedua saudara saya Arsi Rahma Purnamasari dan Ghaly Ahmad Barkah. Serta Taqiyah Fitriyani yang telah memberikan semangat dan dukungan dari awal sampai akhir pengerjaan skripsi ini.
6. Segenap bapak dan ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada Penulis selama menempuh pendidikan di Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya.
7. Staf administrasi Program Studi Informatika/Ilmu Komputer, Program Teknologi Informasi dan Ilmu Komputer.

Keluarga Besar Mahasiswa Informatika/Ilmu Komputer khususnya angkatan 2011, seluruh teman-teman Kelas H terima kasih atas segala bantuan dan dukungannya selama ini.

Malang, 18 Juli 2018

Faris Muslim Azmi
bigfaris0@gmail.com

ABSTRAK

Perkembangan teknologi informasi mengalami peningkatan dari waktu ke waktu meningkatkan kebutuhan terhadap penggunaan teknologi informasi. Setiap pengguna memiliki kebutuhan layanan informasi yang berbeda-beda. Maka, dibutuhkan suatu sistem *load balancing multi-service* untuk meminimalisir kemacetan *traffic* dan dapat mengurangi beban kerja *server*. *Load balancing* merupakan metode jaringan komputer untuk melakukan pembagian beban pada *server*. *HAproxy* dan *Zevenet* merupakan *load balancer* yang mendukung beragam jenis *service* seperti *HTTP* dan *FTP.* Pada penelitian ini dilakukan perbandingan kinerja *HAproxy* dan *Zevenet* dalam pengimplemetasian *multi-service load balancing*. Pengujian kinerja dilakukan dengan mengirimkan 1000 , 2500 dan 5000 koneksi yang merepresentasikan kondisi jaringan untuk *low, medium* dan *high traffic* pada *service HTTP* dan *FTP*. Algoritme yang digunakan pada pengujian adalah *round robin* dan *least connection* dan parameter yang diuji adalah *response time* dan *resource utilization.* Berdasarkan pengujian yang dilakukan, pada layanan *HTTP* dan *FTP*, *HAproxy* menunjukkan hasil pengujian yang lebih baik pada parameter *response time*. Sedangkan *Zevenet* menunjukkan hasil pengujian yang lebih unggul pada parameter *resource utilization* untuk layanan *HTTP* dan *FTP*.

Kata kunci: *load balancing*, *HTTP*, *FTP, response time, resource utilization*

ABSTRACT

The development of information technology has been growth rapidly over time thus escalated the needs for information services. Each user has different needs of information services. Thus, a multi-service load balancing system is required to minimize traffic congestion and to reduce server's workloads. Load balancing is a computer network method to distribute loads on servers. HAproxy and Zevenet are load balancers that support various services such as HTTP and FTP. In this research, the conducted test is performance comparison of HAproxy and Zevenet in the implementation of multi-service load balancing. Performance testing is done by sending 1000, 2500 and 5000 connections that representated network conditition for low, medium and high traffic on both services HTTP and FTP. The algorithms used in this test are round robin and least connection and the experiments parameter which is used are response time and resource utilization. Based on the performances test result, both on HTTP and FTP services, HAproxy showed better test results on the response time parameter. While Zevenet showed superior test results on resource utilization parameter on both HTTP and FTP services.

Keywords: load balancing, HTTP, FTP, response time, resource utilization

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Perkembangan teknologi informasi mengalami peningkatan dari waktu ke waktu meningkatkan kebutuhan terhadap penggunaan teknologi informasi. Instansi dan industri maupun pelaku bisnis menggunakan teknologi informasi untuk memberikan pelayanan yang lebih baik dan meningkatkan produktifitas serta untuk memudahkan dalam mengatasi permasalahan yang dihadapi. Bahkan, masyarakat saat ini tidak lepas dari kebutuhan terhadap informasi dalam kegiatan sehari-hari. Akses internet yang mudah menjadi pemicu terhadap peningkatan kebutuhan informasi tersebut (Chauhan, 2012). Meningkatnya *request* pada layanan penyedia informasi menuntut akses yang cepat untuk mendapatkan informasi. Peningkatan *request* tersebut berdampak pada penambahan beban kerja pada *server*, yang berfungsi sebagai aktor yang menangani *request* dan media penyimpan informasi.

Maka, dibutuhkan suatu sistem yang dapat meminimalisir kemacetan *traffic* data dan mengurangi beban kerja server. Solusi untuk mengatasi permasalahan tersebut adalah dengan melakukan *load balancing*. *Load balancing* bekerja dengan mendistribusikan *request* pada sejumlah server-server terdistribusi secara cepat dan merata sehingga mengurangi beban kerja server serta meningkatkan kapasitas dan reliabilitas dari server tersebut (IBM, 2017). Setiap pengguna memiliki kebutuhan layanan informasi yang berbeda-beda. Layanan informasi tidak terbatas pada layanan berbasis web, terdapat kebutuhan terhadap layanan *file sharing* seperti *FTP service*. Perbedaan jenis *service* berdampak pada perbedaan waktu penyelesaian dan beban kerja *server* yang berbeda.

Perangkat lunak yang menyediakan layanan penyeimbang beban dengan fitur yang mendukung beragam jenis *service* seperti *HTTP service* dan *FTP service* adalah *HAproxy* dan *Zevenet*. Kedua perangkat lunak tersebut bersifat *open-source*, sehingga dapat digunakan oleh siapa saja dan dimana saja. Adapun perbedaan kedua perangkat lunak tersebut adalah *HAproxy* menggunakan metode *reverse proxy* dalam melakukan penyeimbangan beban (HAproxy, 2017). Sedangkan *Zevenet* merupakan sebuah *load balancer* yang berbasis *Lx4NAT* (Zevenet, 2017). Sehingga, dengan melakukan implementasi dan pengujian kinerja *Haproxy* dan *Zevenet* pada layanan *multi service* dapat menentukan *load balancer* yang dapat meminimalisir kemacetan *traffic* data serta mengurangi beban kerja server. Mengacu pada permasalahan yang telah disampaikan, judul yang diambil dalam skripsi ini adalah "**Perbandingan Kinerja *HAproxy* Dan *Zevenet* Dalam Pengimplemetasian *Multi Service Load Balancing***". Diharapkan topik skripsi yang diangkat dapat memberikan solusi dari permasalahan yang telah disampaikan.

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dijabarkan, maka dapat dirumuskan beberapa permasalahan sebagai berikut:

1. Bagaimana pengimplementasian *multi service load balancing* menggunakan *HAproxy* dan *Zev*enet?
2. Bagaimana perbandingan kinerja *HAproxy* dan *Zevenet* dalam pengimplementasian *multi service load balancing*?

1.3 Tujuan

Tujuan dari penulisan skripsi ini adalah :

1. Pengimplementasian *multi service Load Balancing* menggunakan *HAproxy* dan *Zevenet*.
2. Menganalisa perbandingan kinerja *HAproxy* dan *Zevenet* dalam pengimplementasian *multi service load balancing*.

1.4 Manfaat

Manfaat penelitian ini bagi instansi yang menyediakan layanan informasi, khususnya di Universitas Brawijaya. Penelitian ini dapat membantu dan dijadikan rujukan oleh *system administrator* untuk mengetahui hasil kinerja dari *HAproxy* dan *Zevenet* pada kluster *server multi-service*. Dengan dilakukan penelitian ini diharapkan dapat meminimalisir kemacetan *traffic data* dan mengurangi beban kerja *server*, sehingga dapat mengoptimalkan penyediaan informasi. Selain itu, penelitian ini dapat menjadi menjadi acuan penelitian berikutnya terkait dengan topik *load balancing multi service* untuk dapat diterapkan dalam berbagai permasalahan.

1.5 Batasan masalah

Dalam batasan masalah yang dihadapi diperlukan ruang lingkup permasalahan terhadap perbandingan kinerja HAproxy dan Zevenet dalam pengimplemetasian multi service load balancing. Hal ini dilakukan dengan tujuan agar pembahasan masalah tidak terlalu meluas. Maka batasan ruang lingkup masalah dibahas sebagai berikut:

1. Implementasi *server* dan *load balancer* dilakukan pada lingkungan *virtual*.
2. *Service* yang digunakan meliputi *HTTP service dan FTP service*.
3. Parameter yang dianalisa adalah *response time* dan *resource utilization*.
4. *Server* penyimpanan dan sinkronisasi data menggunakan metode *network file systems (NFS).*
5. Tidak membahas aspek keamanan jaringan pada sistem yang dibangun.

1.6 Sistematika pembahasan

Sistematika pembahasan penelitian yang disusun ini akan dibahas pada bab-bab yang akan diuraikan seperti dibawah ini:

BAB 1: PENDAHULUAN

Dalam bab ini diuraikan tentang latar belakang permasalahan mencoba merumuskan inti permasalahan dan menentukan tujuan untuk kegunaan penelitian yang kemudian diikuti dengan manfaat, pembatasan masalah, serta sistematika pembahasan.

BAB 2: LANDASAN KEPUSTAKAAN

Bab ini berisi semua dasar-dasar teori serta kepustakaan untuk selanjutnya digunakan pada bagian perancangan, implementasi, pengujian dan analisis .

BAB 3: METODELOGI

Bab ini membahas tentang metode yang digunakan diantara studi literatur, analisa kebutuhan sistem, perancangan dan implementasi sistem serta pengujian dan analisis perbandingan kinerja *HAproxy* dan *Zevenet* dalam pengimplementasian *multi service load balancing*.

BAB 4: PERANCANGAN DAN IMPLEMENTASI

Bab ini berisi tentang perancangan dan implementasi *HAproxy* Dan *Zevenet* dalam pengimplemetasian *multi service load balancing*.

BAB 5: PENGUJIAN DAN ANALISIS HASIL

Bab ini berisi tentang pengujian, pengambilan data serta analisa perbandingan kinerja *HAproxy* dan *Zevenet* dalam pengimplementasian *multi-service load balancing*.

BAB 6: PENUTUP

Bab ini berisi kesimpulan dari perbandingan kinerja *HAproxy* dan *Zevenet* dalam pengimplemetasian *multi service load balancing.* Diberikan juga saran-saran yang dapat dijadikan acuan untuk penelitian berikutnya.

BAB 2 LANDASAN KEPUSTAKAAN

1.1 Penelitian Terdahulu

Sebagai bahan pertimbangan dalam peneliltian ini, maka dicantumkan beberapa hasil penelitian terdahulu oleh beberapa peneliti. Berikut tabel perbandingan antara penelitian terdahulu dengan penelitian yang dilakukan:

Tabel 2. 1 Perbandingan Penelitian

No.	Nama Penulis,Tahun dan Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1	Yogesh Chauhan dan Siva Chauhan, 2012. *Performance Comparison of Server Load Distribution With FTP and HTTP (Chauhan, 2012).*	Implementasi dan analisa kinerja load balancer pada *service HTTP* dan *FTP* menggunakan algoritme *round-robin* dan *least connections*	Menggunakan *OPNet* sebagai *load balancer* untuk distribusi data dengan parameter *resource utilization.*	Menggunakan *Load balancer HAproxy* dan *Zevenet* pada *service HTTP* dan *FTP*. Parameter pengujian *response time* dan *resource utilization.* Algoritme *load balancing round robin* dan *least connection.*
2	Sandeep Sharma, Sarabjit Singh, 2008, *Performance Analysis of Load Balancing Algorithms (Sharma, 2008)*	Implementasi dan analisa kinerja *load balancing* dengan algoritme *round robin* pada *service HTTP*	Analisa kinerja algoritme *load balancing local queue algorithm, threshold algorithm* pada *service HTTP* dengan parameter *fault tolerant, over load rejection.*	Menggunakan *Load balancer HAproxy* dan *Zevenet* pada *service HTTP* dan *FTP*. Parameter pengujian *response time* dan *resource utilization.* Algoritme *load balancing round robin* dan *least connection.*

2.1 Landasan Teori

Landasan teori adalah seperangkat definisi, konsep serta proposisi yang telah disusun rapi serta sistematis tentang variabel-variabel dalam sebuah penelitian. Landasan teori menjadi dasar yang kuat dalam penelitian ini.

2.1.1 Load Balancing

Load Balancing adalah sebua proses dan teknologi untuk *traffic* sebuah situs pada beberapa *server* menggunakan perangkat berbasis jaringan ataupun *software*. *Load balancer* mendistribusikan *request* klien yang dikirim menuju server secara dinamis dan seragam di semua *server* yang tersedia. Layanan *Load Balancing* memungkinkan pengaksesan sumber daya dalam jaringan didistribusikan ke beberapa *host* lainnya agar tidak terpusat sehingga unjuk kerja jaringan komputer secara keseluruhan bisa stabil (Bourke, 2001).

Ketika sebuah sebuah *server* sedang diakses oleh para pengguna, maka sebenarnya *server* tersebut sebenarnya sedang terbebani karena harus melakukan proses permintaan kepada para penggunanya. Jika penggunanya banyak maka prosesnya akan banyak. Jika satu *server* saja terbebani, tentu *server* tersebut tidak bisa banyak melayani para penggunanya karena kemampuan melakukan *processing* ada batasnya (IBM, 2001). Solusi yang paling ideal adalah dengan membagi-bagi beban yang datang ke beberapa *server*. Jadi yang melayani pengguna tidak hanya terpusat pada satu perangkat saja.

2.2.1.1 Tipe Load Balancing

Dalam dunia *load balancing*, ada dua pilihan untuk dipertimbangkan ketika merancang solusi *load balancing*. Pilihan solusinya adalah menggunakan *software Load Balancing* atau *Hardware Load Balancing*. Setiap pilihan memiliki persyaratan, kelebihan, dan kelemahan tersendiri. Dan dari tipenya *load balancing* dapat dibedakan menjadi 2 tipe, yaitu:

a) *Software Load Balancing*. Dimana *load balancing* berjalan disebuah *PC/Server*, dan aplikasi *load balancing* di install dan perlu dikonfigurasi sebelum dapat berfungsi. Keuntungannya adalah jika ada penambahan fitur atau fasilitas tambahan tidak perlu mengganti keseluruhan perangkat *load balancing*.
b) *Hardware Load Balancing*. Dimana *load balancing* berjalan disebuah *device*/alat yang sudah disiapkan dari pabrik dan siap digunakan. Tipe *Hardware Load Balancing* banyak digunakan karena kemudahannya. (Kopparapu, 2002).

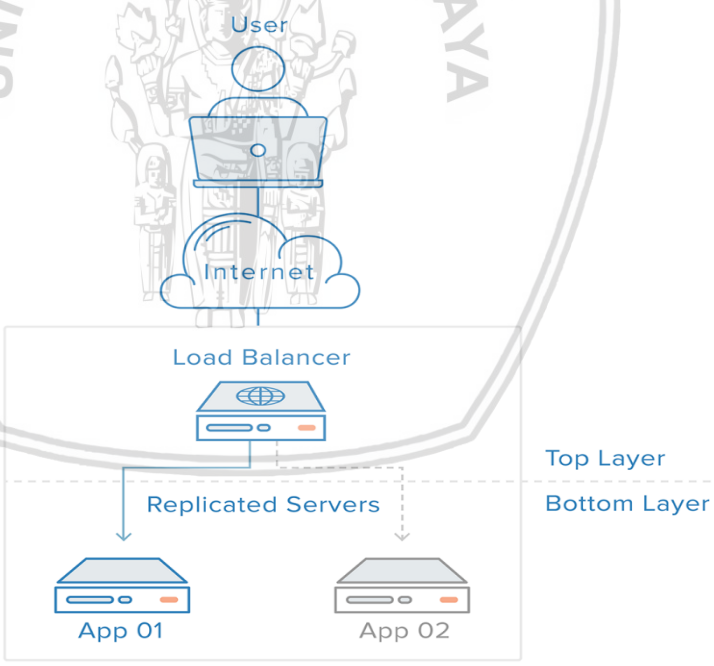
2.2.1.2 Fungsi Load Balancing

Beberapa fitur yang ada pada baik *load balancer* berbasis *hardware* maupun *load balancer* berbasis *software*, yaitu:

a.) Memotong *traffic* berbasis jaringan *(seperti traffic web) yang ditujukan untuk sebuah situs*.
b.) Membagi *traffic* menjadi permintaan individu dan memutuskan *server* mana yang menerima permintaan tersebut.
c.) Melakukan *monitoring* pada *server* yang tersedia, memastikan *server* tersebut sanggup merespons *traffic* yang masuk.
d.) Menyediakan redundansi dengan menggunakan lebih dari satu unit dalam skenario *failover* (Bourke, 2001).

2.2.1.3 Arsitektur Load Balancing

Load balancing adalah metode yang paling mudah untuk menskalakan infrastruktur *server* aplikasi. Seiring meningkatnya permintaan aplikasi, *server* baru dapat dengan mudah ditambahkan ke kluster *server backend*, dan *load balancer* akan segera mulai mengirimkan *traffic* data ke *server* baru. Biasanya, semua *server backend* akan memasok konten yang sama sehingga pengguna menerima konten yang konsisten terlepas dari *server* mana yang meresponsnya.



Gambar 2. 1 Topologi *Load Balancing*

Pada contoh di atas, pengguna mengakses *load balancer*, yang meneruskan permintaan pengguna ke *server backend*, yang kemudian merespons langsung permintaan pengguna (Kopparapu, 2002).

2.2.1.4 Algoritme Load Balancing

Load balancing yang efektif secara cerdas menentukan *server* yang paling baik dalam *cluster server* tertentu yang dapat memproses paket data masuk. Dengan melakukan itu, diperlukan algoritme yang diprogram untuk mendistribusikan beban *traffic* data dengan cara tertentu. Ada banyak algoritme yang dapat digunakan untuk secara cerdas memuat permintaan akses klien yang seimbang di seluruh *cluster server*. Algoritme sangat bervariasi, tergantung pada apakah suatu beban didistribusikan pada jaringan atau lapisan aplikasi (Bourke, 2001).

Algoritme *load balancing* mempengaruhi efektivitas mekanisme distribusi beban, kinerja dan kelangsungan bisnis. Berikut adalah sejumlah algoritme *load balancing* yang populer digunakan:

a.) Round Robin

Algoritme *Round Robin* merupakan algoritme yang paling sederhana dan banyak digunakan oleh perangkat *load balancing*. Algoritme ini membagi beban secara bergiliran dan berurutan dari satu *server* ke *server* lain sehingga membentuk putaran.

b.) *Least Connection*.

Algoritme *Least Connection* akan melakukan pembagian beban berdasarkan banyaknya koneksi yang sedang dilayani oleh sebuah *server*. *Server* dengan pelayanan koneksi yang paling sedikit akan diberikan beban yang berikutnya akan masuk. Namun, algoritme ini juga mendasarkan pilihan mereka pada kapasitas *server* (Hunt, 2002).

2.1.1 HAproxy

HAproxy adalah singkatan dari *High Availability Proxy* produk *open-source* yg mendukung keperluan *load balancing* untuk *TCP/HTTP*. Haproxy ditulis dalam bahasa pemrograman *C* dan berjalan pada sistem operasi berbasis *Linux* dan *FreeBSD*. Penggunaannya yang paling umum adalah memperbaiki kinerja dan keandalan lingkungan *server* dengan mendistribusikan beban kerja ke beberapa *server*. *HAproxy* adalah *load balancer* yang bekerja pada lapisan aplikasi (*Layer 7*) *OSI* dan merupakan solusi yang digunakan untuk menerapkan *reverse proxy* untuk layanan internet berbasis *HTTP* dan *TCP* (HAproxy, 2017).

HAproxy mendistribusikan beban kerja pada satu kluster *server* untuk memaksimalkan kinerja dan mengoptimalkan sumber daya. Beban aplikasi pada *front-end server* yang sangat bergantung pada *back-end database* dapat dengan mudah didistribusikan walau dengan banyaknya koneksi yang berjalan secara bersamaan. Semua klien akan terhubung kebagian server penyeimbang beban dan *proxy* akan meneruskan ke salah satu *server* basis data yang tersedia berdasarkan skema *load balancing* yang digunakan (HAproxy, 2017)

2.2.2.1 Fitur HAproxy

Pada *HAproxy* memiliki beberapa fitur-fitur unggulan di dalamnya yaitu sebagai berikut:

a.) Metode *Load Balancing* Komprehensif. Metode atau algoritme load balancing dasar dan lanjutan termasuk *round-robin statis* dan *weighted*, *weighteds statis* dan *weighted*, *client IP* atau *URI-HASH*.

b.) Stickiness/Persistence . Beberapa pilihan persistensi berdasarkan informasi *TCP/IP (IP klien, port* atau *TCP* muatan) atau properti permintaan *HTTP (cookies, header, URI)*

c.) *Logging*. Kemampuan *logging* yang canggih untuk membangun jalur *log* yang dapat disesuaikan sepenuhnya yang memungkinkan untuk mengetahui dengan tepat apa yang sedang terjadi pada sistem (Haproxy, 2017).

2.1.1 Zevenet

Zevenet atau yang dulu dikenal dengan *Zen Load Balancer* adalah proyek *open source* yang memungkinkan untuk membuat sistem terdistribusi yang terukur dan terukur. *Zevenet* didistribusikan di bawah format *ISO* standar yang ada di atas distribusi *GNU/ Debian Linux* yang umum. *Zevenet* adalah salah satu solusi terbaik untuk load balancing *TCP* dan *UDP* untuk menyediakan ketersediaan sistem terdistribusi tinggi. *Zevenet* bukanlah *gateway* atau *firewall*, perangkat lunak ini adalah penyeimbang beban dengan fungsi lanjutan untuk diintegrasikan dan dikonsolidasikan dalam layanan produksi yang ada. Ini berjalan di *Linux* dan mendukung hampir semua konfigurasi *hardware* fisik atau *virtual*. Penggunaan umum untuk *Zevenet* mencakup *TCP Simple Load Balancer* yang dapat bekerja dengan layanan seperti *SMTP, LDAP, FTP, IMAP, POP, HTTP*, atau layanan lainnya melalui *protokol TCP*. (Zevenet, 2017).

2.2.3.1 Fitur Zevenet

Zevenet menyediakan fungsionalitas canggih untuk menawarkan layanan high availibility dan load balancing seperti berikut:

a.) *Advanced Load Balancing.* Persistensi *routing Layer 3*. *Layer 7 cookies*, persistensi sesi *header* dan *ip*. Pemeriksaan lanjutan untuk pemantauan *backend*. *UDP, TCP* dan beberapa penyedia *uplink*. Profil lanjutan untuk layanan *HTTP* dan *HTTPS*.

b.) Konfigurasi kluster tak terbatas. Konfigurasi *backend* tak terbatas. Metode *SNAT* pada *load balancing layer 7*.

c.) *Multi-protocols*. *HTTP, HTTPs, ssh, pop3, imap, smtp, dns, ntp, ldap, ldaps, rdp* dan layanan *TCP/UDP* lainnya (Zevenet, 2017).

2.1.1 HTTP

Layanan *HTTP* adalah komponen dari *Application Server* yang menyediakan fasilitas untuk menyebarkan aplikasi web dan untuk membuat aplikasi web yang dikerahkan yang dapat diakses oleh klien *HTTP* seperti *SOAP* dan *REST*. *HTTP* berarti *HyperText Transfer Protocol*. *HTTP* adalah protokol dasar yang digunakan oleh *World Wide Web* dan protokol ini mendefinisikan bagaimana pesan diformat dan dikirimkan, dan tindakan apa yang harus dilakukan oleh *server* dan *browser Web* dalam menanggapi berbagai perintah. Kelemahan *HTTP* ini ditangani menggunakan sejumlah teknologi baru, termasuk *ActiveX, Java, JavaScript dan cookies*. Layanan *HTTP* memungkinkan aplikasi *server* menerima permintaan *HTTP*, mengirim tanggapan *HTTP,* dan menanggapi *HTTP cache* di *kernel*. Aplikasi *server* dapat mengirim dan menerima koneksi *HTTP* atau *HTTPS (Secure Sockets Layer/SSL)*.

Sesi *HTTP* adalah urutan transaksi permintaan-respons jaringan. Klien *HTTP* memulai permintaan dengan membuat koneksi *Transmission Control Protocol (TCP)* ke port tertentu pada *server* (biasanya *port* 80, kadang-kadang *port* 8080). *Server HTTP* yang mendengarkan di *port* tersebut menunggu pesan permintaan klien. Setelah menerima permintaan tersebut, *server* mengirimkan kembali baris status, seperti "*HTTP / 1.1 200 OK*", dan pesannya sendiri. *HTTP* menyediakan beberapa skema otentikasi seperti otentikasi akses dasar yang beroperasi melalui mekanisme respons dimana *server* mengidentifikasi sebelum melayani konten yang diminta. Setiap klien dapat menggunakan metode apapun dan *server* dapat dikonfigurasi untuk mendukung kombinasi metode apapun. Jika metode tidak diketahui, metode tersebut akan diperlakukan sebagai metode yang tidak aman (Hunt, 2002).

Tabel 2. 2 Method Protokol *HTTP*

Method	Deskripsi	*Body*
GET	Ambil dokumen dari *server*	Tidak
HEAD	Ambil *header* dokumen dari *server*	Tidak
POST	Kirimkan data ke *server* untuk diproses	Ada
PUT	Simpan data yang ada di bagian *Body* ke *server*	Ada
TRACE	Ikuti jejak pesan dari *proxy server* sampai ke *server*	Tidak
OPTIONS	Temukan *method* apa saja yang dapat dijalankan oleh *server*	Tidak
DELETE	Hapus data dari *server*	Tidak

2.1.1 FTP

Layanan *FTP* adalah sebuah layanan yang menjalankan protocol *FTP* yang merupakan protokol untuk pertukaran *file* melalui Internet. *File Transfer Protocol* (*FTP*) adalah protokol yang umum digunakan untuk bertukar file melalui Internet. *FTP* menggunakan protokol *TCP / IP* Internet untuk memungkinkan transfer data. *FTP* menggunakan arsitektur client-server, sering diamankan dengan *SSL / TLS*. *FTP* mempromosikan pembagian file melalui komputer jarak jauh dengan transfer data yang andal dan efisien. *FTP* juga bisa dikatakan sebuah protokol Internet yang berjalan di dalam lapisan aplikasi yang merupakan standar untuk pentransferan berkas (*file*) komputer antar mesin-mesin dalam *framework*. *FTP* dapat berjalan dalam mode aktif atau pasif, yang menentukan bagaimana koneksi data dibuat. Dalam kedua kasus tersebut, klien membuat koneksi kontrol *TCP* dari *port* N yang acak, biasanya tidak berpengalaman, ke port perintah *server FTP* 21.

Login FTP menggunakan skema *username* dan *password* biasa untuk memberikan akses. *Username* dikirim ke *server* menggunakan perintah *USER*, dan kata kunci dikirim menggunakan perintah *PASS*. Jika *server* mendukungnya, pengguna dapat masuk tanpa memberikan kredensial masuk, namun *server* yang sama hanya mengizinkan akses terbatas untuk sesi semacam itu. *FTP* biasanya mentransfer data dengan meminta *server* terhubung kembali ke klien, setelah perintah *PORT* dikirim oleh klien. Ini bermasalah untuk kedua *NAT* dan *firewall*, yang tidak memungkinkan koneksi dari Internet ke *host* internal. Untuk *NAT,* komplikasi tambahan adalah bahwa representasi alamat *IP* dan nomor *port* dalam perintah *PORT* merujuk ke alamat *IP host internal* dan *port*, bukan alamat *IP publik* dan *port NAT* (Hunt, 2002).

FTP melalui *SSH* adalah praktik *tunneling* sesi *FTP* normal melalui koneksi *Secure Shell*. Karena *FTP* menggunakan beberapa koneksi *TCP* (tidak biasa untuk protokol *TCP/IP* yang masih digunakan), sangat sulit untuk melakukan terowongan melalui *SSH*. Dengan banyak klien *SSH,* mencoba memasang terowongan untuk saluran kontrol (koneksi *client-to-server* awal pada *port* 21) hanya akan melindungi saluran itu. Ketika data ditransfer, perangkat lunak *FTP* di kedua ujungnya menyiapkan koneksi *TCP* baru (saluran data) dan karenanya tidak memiliki kerahasiaan atau perlindungan integritas. *FTPS* adalah perluasan dari standar *FTP* yang memungkinkan klien meminta sesi *FTP* untuk dienkripsi. Hal ini dilakukan dengan mengirimkan perintah "*AUTH TLS*". Server memiliki pilihan untuk mengizinkan atau menolak koneksi yang tidak meminta *TLS* (RedHat, 2017).

BAB 3 METODELOGI PENELITIAN

Metodelogi penelitian merupakan pembahasan metode yang digunakan untuk perancangan, implementasi, pengujian dan analisa kinerja *HAproxy* dan *Zevenet* pada pengimplementasian *multi service load balancing*. Metodelogi penelitian dalam penelitian ini meliputi perumusan masalah, perancangan sistem, implementasi sistem, pengujian sistem, analisis hasil performa sistem dan terakhir pengambilan kesimpulan. Penelitian bertipe deskriptif, dengan kata lain mengimplementasikan, menguji dan analisa serta mendeskripsikan perbandingan kinerja *HAproxy* dan *Zevenet* pada pengimplementasian *multi-service load balancing*. Pada **Gambar 3.1** diagram alir metode penelitian dimulai dari merumuskan masalah, melakukan perancangan, kemudian melakukan implementasi, percobaan pengujian, analisis hasil pengujian dan terakhir pengambilan kesimpulan.

Gambar 3. 1 Diagram Alir Metode Penelitian

3.1 Studi Literatur

Tahap ini merupakan tahap pertama pengembangan dari analisis yang dilakukan. Studi pustaka dilakukan untuk mencari sumber referensi dan penelitian sebelumnya yang terkait. Referensi dapat berupa berkala ilmiah, buku teks, skripsi, ataupun dokumentasi hasil penelitian lainnya dimana referensi tersebut memiliki keterkaitan dengan perbandingan kinerja *HAproxy* dan *Zevenet* dalam pengimplemetasian *multi service load balancing* sebagai referensi dalam penelitian ini.

3.2 Analisis Kebutuhan

Analisis kebutuhan dilakukan dengan tujuan untuk mengetahui kebutuhan yang diperlukan terkait dengan perbandingan kinerja *HAproxy* dan *Zevenet* dalam pengimplemetasian *multi service load balancing*. Untuk mempermudah analisa kebutuhan sistem dalam menentukan keseluruhan kebutuhan secara lengkap, maka dibagi kebutuhan sistem menjadi dua jenis, yaitu kebutuhan fungsional dan kebutuhan non fungsional.

3.2.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan kebutuhan yang berisi tentang pernyataan proses-proses yang dilakukan oleh sistem. Proses-proses ini terdiri dari layanan yang harus disediakan oleh sistem, bagaimana sistem bereaksi pada input tertentu dan bagaimana perilaku sistem pada situasi tertentu. Berikut adalah kebutuhan fungsional sistem:

1. *HAproxy* dan *Zevenet* mampu menghubungkan *client* pada *HTTP service.*
2. *HAproxy* dan *Zevenet* mampu menghubungkan *client* pada *FTP service.*
1. *HAproxy* dan *Zevenet* mampu melakukan pendistribusian *request* kepada setiap *server*.
2. Setiap *server* mampu terhubung dan meng-sinkronisasikan data pada *server NFS*

3.2.2 Kebutuhan Non-Fungsional

Kebutuhan non-fungsional dapat dikatakan sebagai jenis kebutuhan yang berisi sifat dan lingkungan yang dimiliki oleh sistem. Adapun sifat yang dimiliki oleh sistem meliputi:

1. Setiap *server* memiliki *private IP address* sehingga tidak bisa diakses dari jaringan luar.
2. *Load balancer* memilah *server* mana yang diteruskan *request* dari *client* berdasarkan algoritme *round robin* dan *least connections*.

Sedangkan kebutuhan *environment* sistem berupa *hardware* dan *software* yang digunakan :

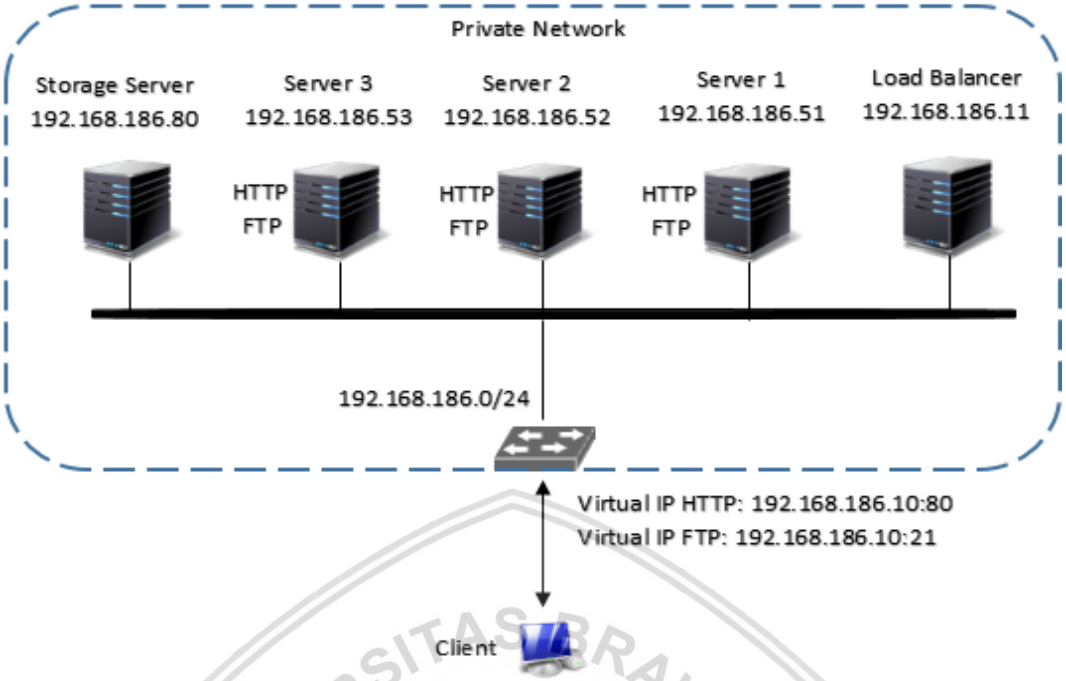
1. Satu unit komputer dengan sistem operasi *Windows*.
2. Aplikasi *Virtualbox* untuk membuat *virtual machine.*
3. Sistem operasi *Ubuntu Server* sebagai *server.*
4. Aplikasi *Load balancer HAproxy*.
5. Aplikasi *Load balancer Zevenet Community edition.*
6. Aplikasi *Apache2 Web Server* untuk *web service.*
7. Aplikasi *VSFTPD* yang berfungsi sebagai *FTP service.*
8. Aplikasi *NFS-Kernel-Server* yang berfungsi sebagai *NFS server.*
9. Aplikasi *Apache JMeter* untuk mengirimkan *request* dan pengambilan data.
10. Aplikasi *Wireshark* untuk men-*capture* paket data dan menganalisa trafik.

3.3 Perancangan

Pada perancancangan sistem, dijelaskan perancangan yang menggambarkan arsitektur jaringan dan konfigurasi sistem yang dibangun secara garis besar. Terdapat dua model jaringan, yaitu jaringan *Public* dan jaringan *Private*. *Public network* merupakan jaringan eksternal yang terdapat satu buah komputer *client*. Sedangkan pada jaringan *private* terdapat sebuah kluster yang memiliki tiga buah *server* dan sebuah *server storage* dan *load balancer*. Pada *server* 1, *server* 2, *server* 3 dijalankan *multi-service* yakni *HTTP service* dan *FTP service*. Seluruh *server* dihubungkan pada *server NFS* yang berfungsi untuk mensinkronisasi data dan penyimpanan data.

Terdapat *load balancer HAproxy* atau *Zevenet* yang terhubung dengan *server multi-service* yang berperan sebagai penerima *request* dari *client* dan mendistribusikan setiap *request* tersebut ke seluruh *server-server multi-service* berdasarkan algoritme *load balancing* yang digunakan yakni algoritme *round robin* dan *leasts connection*. Pada *load balancer* tersebut terdapat sebuah alamat *virtual IP* dengan nomor *port* berbeda yang digunakan sesuai dengan *service* yang dijalankan oleh *server-server*.

Pada *service HTTP* akan digunakan alamat *virtual IP* 192.168.186.10 *load balancer* dengan nomor *port* tujuan 80 untuk menangani *request HTTP* yang dikirimkan oleh *client,* dan pada *service FTP* akan digunakan alamat *vitual IP* yang sama namun dengan nomor *port* tujuan 21 untuk setiap *request FTP* yang dikirimkan oleh *client*. Dan terakhir pada jaringan publik terdapat sebuah client yang mengirimkan sejumlah request kepada server melalui alamat *Virtual IP load balancer* yang diteruskan oleh *load balancer*, seperti yang tertera pada pada **Gambar 3.2** berikut:



Gambar 3. 2 Desain Topologi Jaringan Sistem

3.4 Implementasi

Implementasi sistem dilakukan dengan memperhatikan pada perancangan sistem yang telah dibuat sebelumnya. Pada tahap ini, sistem yang telah dirancang diimplementasikan pada *environment* pengembangan guna mendapatkan hasil yang diharapkan. Pada setiap *virtual machine* yang menjadi *server multi-service* dimplimentasikan aplikasi *Apache2* untuk *HTTP service*, aplikasi *vsftpd* untuk *FTP service*. Pada *server storage* diimplementasikan aplikasi *NFS-Kernel-Server* untuk proses sinkronisasi dan penyimpanan data. Pada sisi *client* diimplementasikan aplikasi *JMeter* yang berfungsi sebagai pembangkit *request* dan menampilkan status performa *server-server* tersebut, serta aplikasi *Wireshark* untuk *monitoring traffic* data.

Pada *virtual machine* yang menjadi *load balancer* diimplementasikan aplikasi *HAproxy* dan *Zevenet*, kemudian dilakukan konfigurasi agar *load balancer* dapat terhubung dengan masing-masing *server multi-service* agar dapat mendistribusikan *request* yang dikirim oleh *client*. Performa dari *load balancer* sangat dipengaruhi oleh algoritme *load balancing* yang digunakan. Oleh karena itu, tahapan selanjutnya melakukan konfigurasi algoritme *load balancing* yang digunakan. Algoritme *round robin* digunakan untuk menguji perilaku load balancer pada cluster yang terdiri dari server dengan spesifikasi yang identik sedangkan algoritme least connection digunakan untuk menguji load balancer terhadap kluster server pada saat server memiliki kapasitas yang tidak setara.

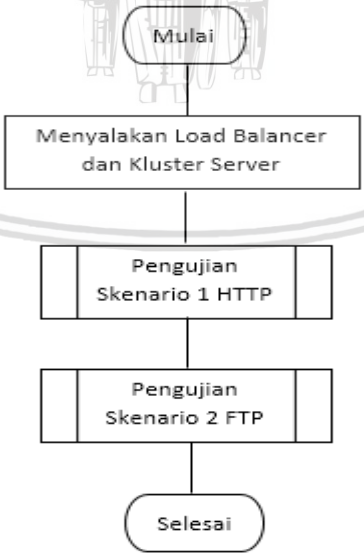
3.5 Pengujian

Pengujian sistem pada penelitian ini dilakukan agar dapat menunjukkan bahwa pengembangan sistem ini telah mampu bekerja sesuai dengan kebutuhan. Selain itu pengujian berguna untuk mendapatkan data dari pengembangan sistem. Untuk pengujian performa digunakan perangkat lunak *Apache Jmeter*. *Apache Jmeter* merupakan perangkat lunak untuk membangkitkan request dan melakukan pengukuran performa pada sebuah *server*. *Apache Jmeter* mensimulasikan beban dengan jumlah request yang banyak pada sebuah *server*. Pada saat pengujian performa terdapat 2 skenario pengujian sistem yang dilakukan. Berikut skenario pengujian:

Tabel 3. 1 Skenario Pengujian

No.	Service	Jumlah Koneksi			Algoritme Yang Diuji		Load Balancer	
1	HTTP	1000	2500	5000	Round-Robin	Least Connections	HAproxy	Zevenet
2	FTP	1000	2500	5000	Round-Robin	Least Connections	Haproxy	Zevenet

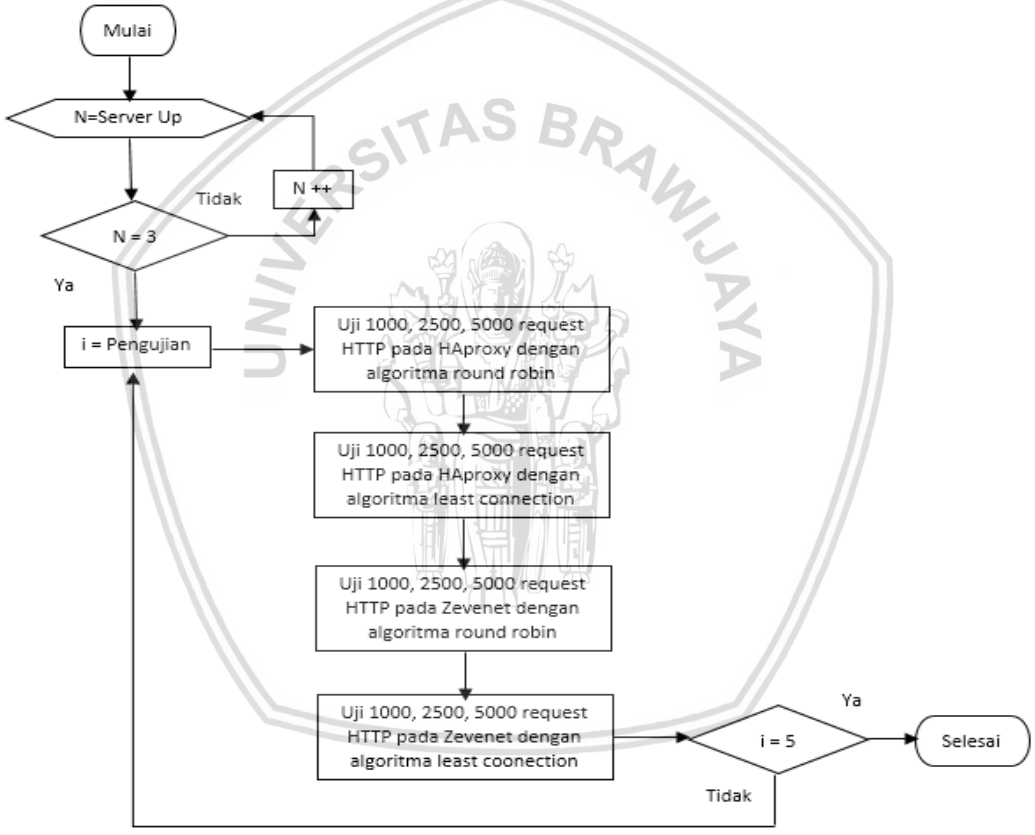
Skenario pengujian dilakukan dengan mengirimkan sebanyak 1000 koneksi untuk merepresentasikan kondisi jaringan saat *low traffic*, 2500 koneksi untuk merepresentasikan kondisi jaringan saat *medium traffic*, dan 5000 koneksi untuk merepresentasikan kondisi jaringan saat *high traffic* pada masing-masing *service* yang berjalan, yakni *HTTP service* dan *FTP service* pada setiap *server* dengan menggunakan algoritme *load balancing round robin* dan *least connections*. Masing-masing proses pengujian dilakukan sebanyak 5 kali. Adapun alur pengujian sistem dijelaskan pada diagram berikut:



Gambar 3. 3 Diagram Alur Pengujian

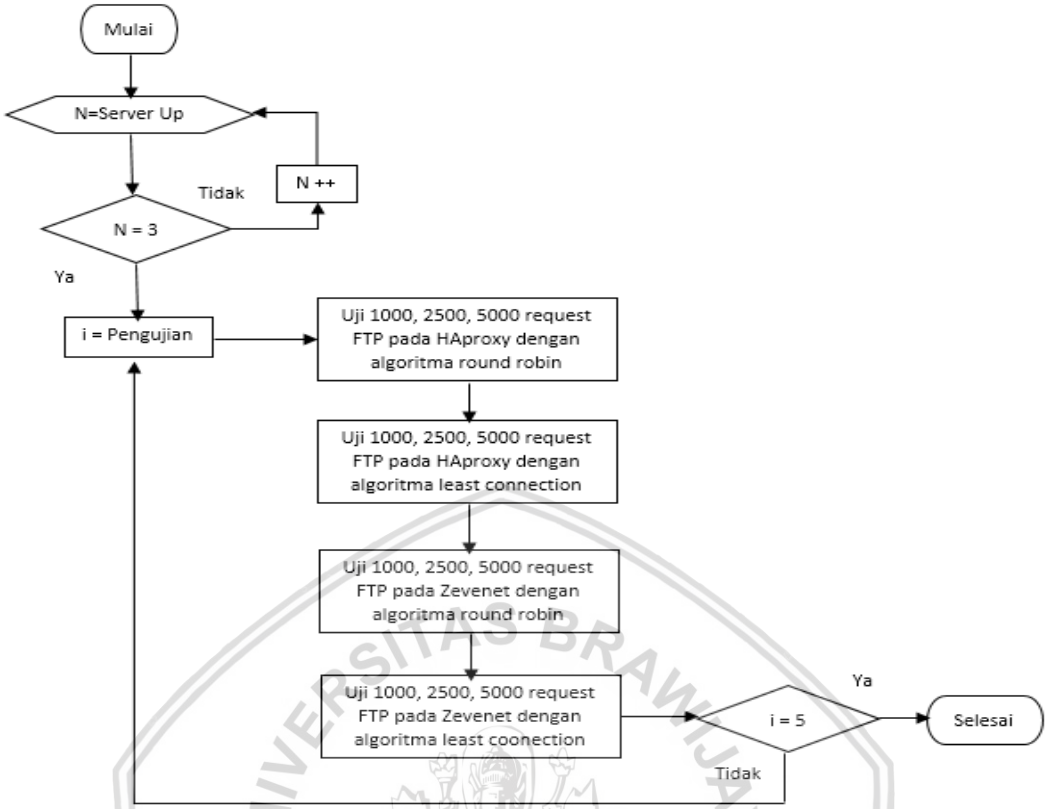
Alur pengujian sistem dimulai dengan menyalakan setiap *server* pada kluster dan *load balancer*. Kemudian melakukan pengujian skenario 1 *HTTP* untuk menguji performa sistem *load balancing HAproxy* dan *Zevenet* pada *service HTTP*. Selanjutnya dilakukan pengujian skenario 2 *FTP* untuk menguji performa sistem *load balancing HAproxy* dan *Zevenet* pada *service FTP*.

Alur pengujian skenario 1 *HTTP* dilakukan dengan cara *client* mengirimkan *request* berdasarkan *service* yang dijalankan oleh *server* dengan jumlah 1000, 5000 dan 10000 koneksi, kemudian *load balancing* akan meneruskan *request* tersebut menuju *server-server* berdasarkan algoritme yang digunakan. Untuk *service HTTP* dilakukan pengujian pada masing-masing *load balancer*, yakni *HAproxy* dan *Zevenet* dengan algoritme *round robin* dan *least connections*. Seperti yang tertera pada **Gambar 3.4** berikut.



Gambar 3. 4 Alur Pengujian Skenario 1 *HTTP*

Alur pengujian skenario 2 FTP dilakukan dengan cara client mengirimkan request berdasarkan *service* yang dijalankan oleh *server* dengan jumlah 1000, 5000 dan 10000 koneksi, kemudian *load balancing* akan meneruskan *request* tersebut menuju *server-server* berdasarkan algoritme yang digunakan. Untuk *service FTP* dilakukan pengujian pada masing-masing *load balancer*, yakni *HAproxy* dan *Zevenet* dengan algoritme *round robin* dan *least connections*. Seperti yang tertera pada **Gambar 3.5** berikut.



Gambar 3. 5 Alur Pengujian Skenario 2 *FTP*

Setiap skenario pengujian dilakukan sebanyak 5 kali perulangan untuk mendapatkan hasil yang optimal dan parameter yang diukur adalah:

1. *Response Time*; Pengukuran parameter ini dilakukan untuk menganalisa waktu tanggap yang dibutuhkan *server* untuk merespon saat paket-paket *request* yang dikirimkan oleh *client* dan juga waktu yang dibutuhkan paket diterima kembali disisi *client*.
2. *Resource Utilization*; Pengukuran parameter ini dilakukan untuk menganalisa seberapa banyak penggunaan sumber daya *CPU* yang digunakan oleh *server* ketika menangani paket-paket request yang dikirimkan oleh *client*.

3.6 Analisis Hasil

Tahap ini menjelaskan analisa kinerja *HAproxy* dan *Zevenet* pada kluster server *multi-service*. Hasil analisa didapatkan melalui perhitungan pengujian kinerja pada parameter yang telah ditentukan, yaitu *response time dan resource utilization* serta skema pengujian fungsional sistem. Diharapkan berdasarkan analisa yang dilakukan dapat diketahui bagaimana hasil implementasi dan analisa kinerja *HAproxy* dan *Zevenet* pada kluster server *multi-service.*

3.7 Kesimpulan

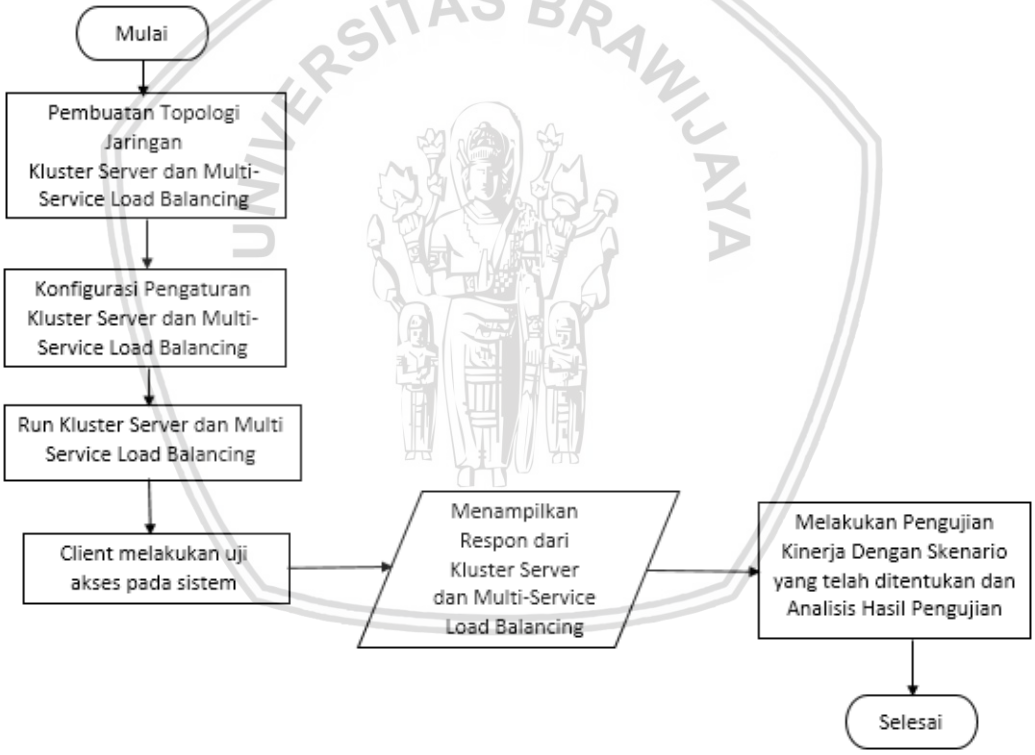
Kesimpulan diambil untuk dijadikan jawaban umum dari perumusan masalah yang telah dibuat sebelumnya. Setelah penarikan kesimpulan secara garis besar, maka diberikan saran-saran yang diharapkan dapat membantu penelitian berikutnya terkait dengan pengembangan *load balancing*.

BAB 4 PERANCANGAN DAN IMPLEMENTASI

Pada bab perancangan dan implementasi ini dibahas lebih rinci mengenai perancangan sistem sampai mengenai cara implementasi sistem sehingga mampu berjalan dengan baik. Perancangan itu sendiri meliputi desain dan alur perbandingan kinerja *HAproxy* dan *Zevenet* dalam pengimplemetasian *multi service load balancing*. Sedangkan implementasi membahas tentang impelementasi *load balancing* dan kluster *server multi-service* serta proses konfigurasi sistem.

4.1 Perancangan Sistem

Pada bagian diagram alir perancangan sistem menjelaskan tentang perancangan sistem yang dimulai dari proses atau langkah-langkah yang tertera pada bab sebelumnya hingga proses pengujian dan analisis hasil yang dilakukan terhadap sistem.



Gambar 4. 1 Diagram Alir Perancangan Sistem

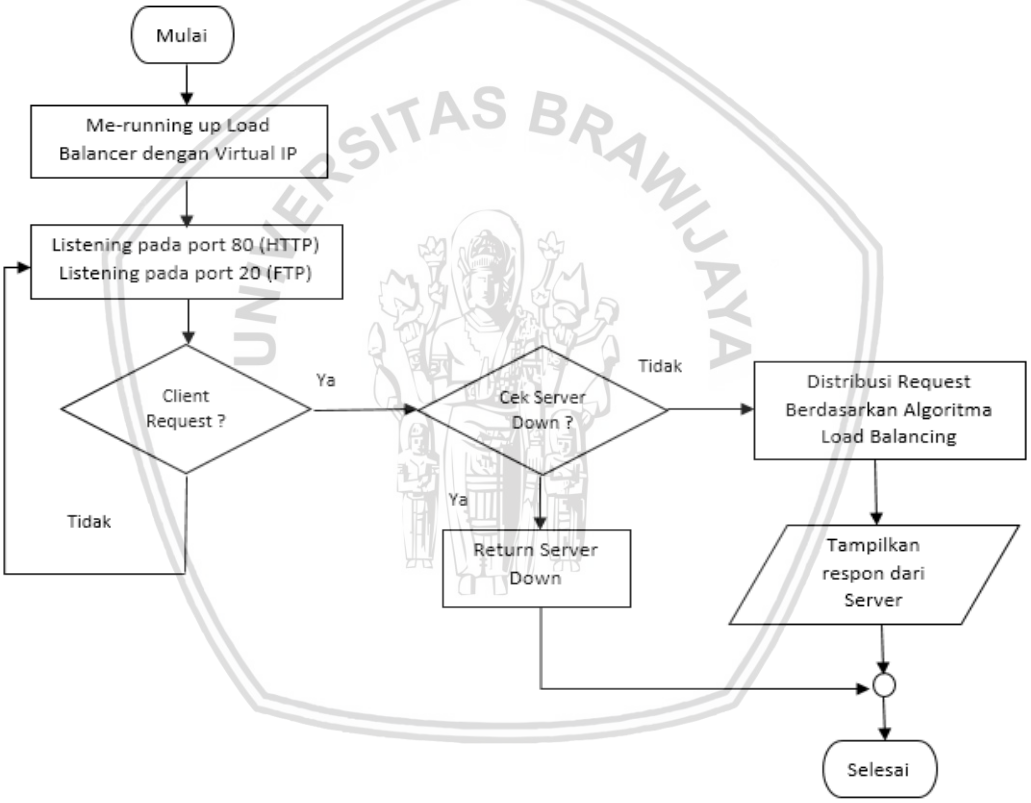
Pada **Gambar 4.1** diatas dapat diketahui bahwa pada tahapan rancangan implementasi sistem dimulai dengan pembuatan topologi jaringan *load balancing* dan kluster *server multi-service*. Setelah itu dilanjutkan dengan proses konfigurasi pengaturan *load balancer* dan *server-server multi-service*. Ketika proses konfigurasi telah selesai kemudian me-*running load balancer* dan *server-server multi service*.

Selanjutnya dilakukan tahapan uji akses dimana *client* mencoba mengirimkan *request* dengan mengakses layanan *server multi-service* melalui *load balancer*

yang mendistribusikan *request* tersebut ke *server-server* pada kluster. Jika client telah mengirimkan *request* ke *server multi-service* maka ditampilkan respon dari *server* yang sebelumnya diteruskan oleh *load balancer* ke client dan menampilkan hasil dari proses pendistribusian *request* oleh *load balancer*. Tahapan terakhir adalah melakukan pengujian kinerja dengan skenario yang telah ditentukan dan melakukan analisis hasil pengujian.

4.1.1 Perancangan *Load Balancing*

Perancangan tahapan *load balancing* menggunakan *load balancer Haproxy* dan *Zevenet* pada kluster *server multi-service* menjelaskan proses-proses yang dilakukan oleh *load balancer* ketika melakukan pendistribusian data yang dikirmkan oleh *client* menuju kluster *server multi-service*. Berikut adalah diagram alir tahapan *load balancing*:



Gambar 4. 2 Diagram Alur *Load Balancing*

Berdasarkan **Gambar 4.2** diatas dapat diketahui tahapan kerja dari proses *load balancing* yang berjalan pada sistem. Tahapan diawali dengan melakukan *running up load balancing* yakni *Haproxy* dan *Zevenet* yang sebelumnya telah dikonfigurasikan agar dapat terhubung dengan kluster *server-server multi-service* dan diberikan alamat *Virtual IP* agar *client* dapat mengakses.

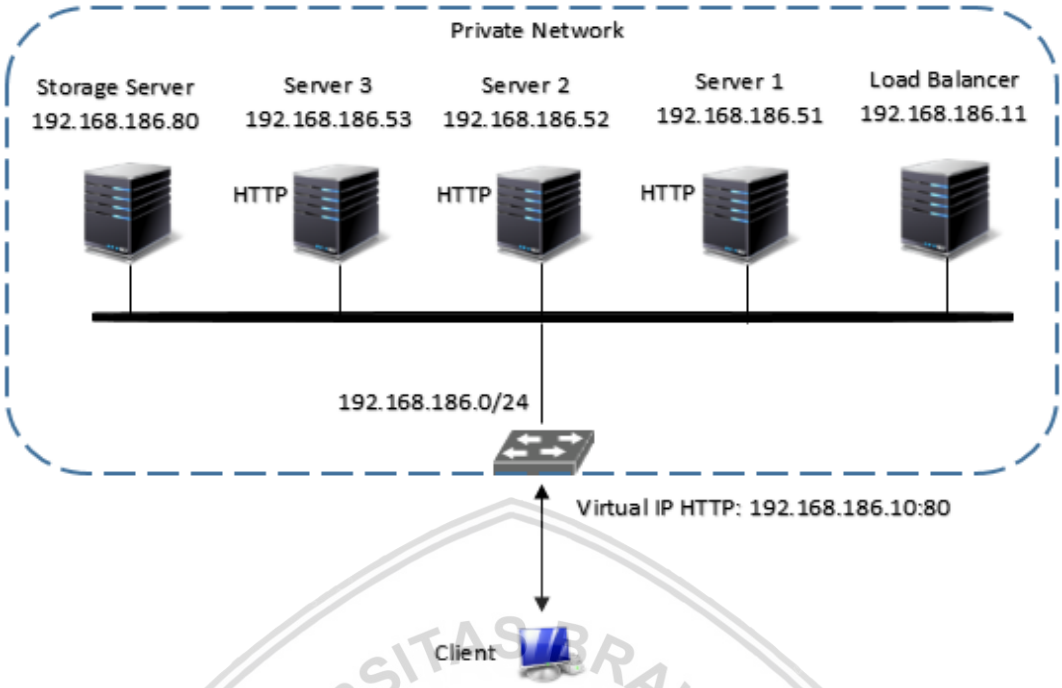
Kemudian *load balancer* mulai *listening* pada masing-masing *port* 80 dan *port* 21. *Port* 80 adalah *port* yang digunakan oleh layanan yang berbasis *HTTP* dan *port* 21 merupakan *port* yang digunakan oleh layanan yang menggunakan protokol berbasis *FTP*. Selanjutnya *load balancer* melakukan pengecekan apabila ada *request* yang masuk. Apabila tidak ada *request* yang masuk, *load balancer* kembali *listening* pada *port-port* yang disebutkan.

Jika terdapat *request* yang masuk melalui *port-port* tersebut, maka selanjutnya *load balancer* mengecek status dari setiap *server multi-service* anggota kluster. Apabila server tersedia, *load balancer* akan melakukan distribusikan request tersebut ke *server multi-service* tujuan dengan melakukan perhitungan berdasarkan algoritme *load balancing* yang digunakan, yakni algoritme *round robin* dan algoritme *least connections*. Apabila *request* yang dikirim berjenis *HTTP*, maka *load balancer* mem-forwarding *request* tersebut pada *port* 80 *server multi-service*. Jika *request* tersebut berjenis *FTP load balancer* meneruskan ke *port* 21 *server multi-service*. Terakhir *load balancer* menerima *respon* dari *server multi-service* dan menampilkannya kembali kepada *client* yang mengirim.

4.1.2 Perancangan *HTTP Service*

Pada perancangan *HTTP service* ini menggunakan *web server clustering* dimana setiap *web server* independen yang bekerja sebagai satu kesatuan sistem. Aplikasi yang digunakan untuk layanan *HTTP* adalah *Apache Web Server* pada masing-masing *server*. *Server* dengan layanan *HTTP* ini menggunakan *port* 80 dalam melakukan proses komunikasi dan transaksi paket berjenis *HTTP* yang diteruskan oleh *load balancer*. Masing-masing *web server* ini terhubung dengan sebuah *server storage NFS* yang telah diinstal aplikasi *NFS* agar dapat melakukan sinkronisasi data sehingga tidak ada terjadi kerancuan data yang diakses oleh setiap *web server*.

Setiap *web server* ini memiliki alamat *IP* yang berbeda, yakni *server* 1 dengan *IP* 192.168.186.51, *server* 2 dengan *IP* 192.168.186.52 dan *server* 3 dengan *IP* 192.168.186.53 dengan *netmask* 255.255.255.0. *Web server* ini terhubung dengan *load balancer Haproxy* dan *Zevenet* yang mendistribusikan paket *request* dari *client*. *Client* mengakses *web server* dengan mengirimkan *request HTTP* melalui *port* 80 *load balancer* dan diteruskan oleh *load balancer* menuju *web server*. *Web server* kemudian memproses *request* tersebut, dan menyambung koneksi dengan *server storage NFS* untuk melakukan proses pengolahan data sesuai dengan isi pesan request yang dikirim oleh *client*. *Server storage NFS* melakukan sinkronisasi data apabila terjadi perubahan pada *file-file* yang disimpan. Kemudian *web server* mengembalikan respon dari *request* yang dikirim kepada *load balancer* melalui *port* 80. Terakhir *load balancer* meneruskan respon dari *web server* menuju alamat *IP client* yang telah mengirim *request*, seperti pada **Gambar 4.3** berikut:

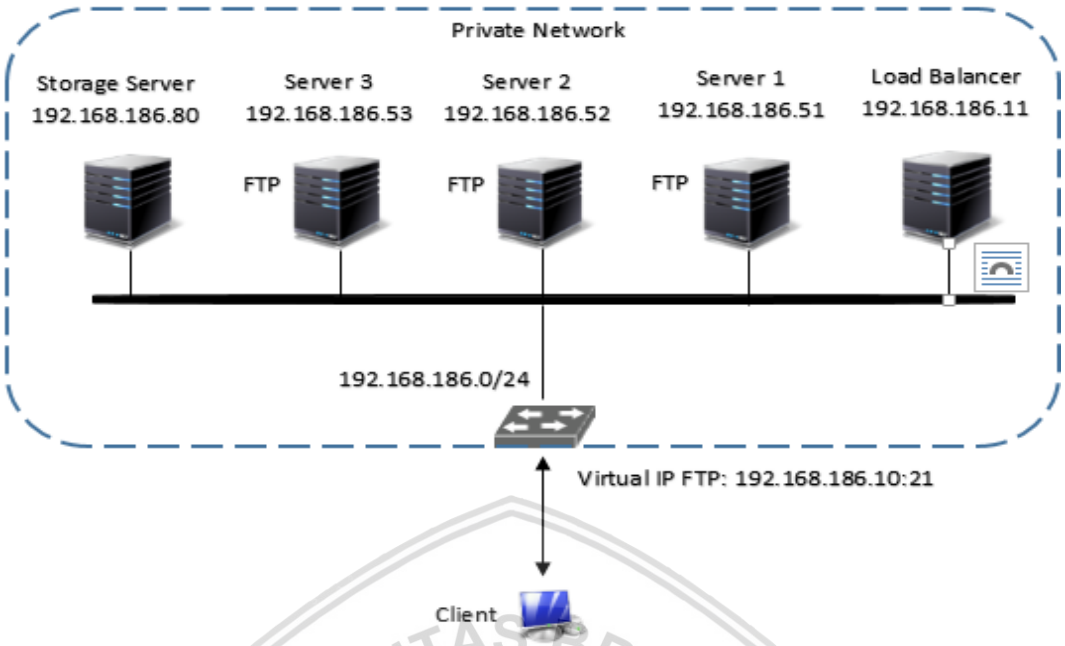


Gambar 4. 3 Topologi *HTTP Service*

4.1.3 Perancangan *FTP Service*

Pada perancangan *FTP service* ini menggunakan *FTP server clustering* dimana setiap *FTP server* independen yang bekerja sebagai satu kesatuan sistem. Aplikasi yang digunakan untuk layanan *FTP* adalah *VSFTPD* pada masing-masing *server*. *Server* dengan layanan *FTP* ini menggunakan *port* 21 dalam melakukan proses komunikasi dan transaksi paket berjenis *FTP* yang diteruskan oleh *load balancer*. Masing-masing *FTP server* ini terhubung dengan sebuah *server storage NFS* yang telah diinstal aplikasi *NFS* agar dapat melakukan sinkronisasi data sehingga tidak ada terjadi kerancuan data yang diakses oleh setiap *FTP server*.

Setiap *FTP server* ini memiliki alamat *IP* yang berbeda, yakni *server* 1 dengan *IP* 192.168.186.51, *server* 2 dengan *IP* 192.168.186.52 dan *server* 3 dengan *IP* 192.168.186.53 dengan *netmask* 255.255.255.0. *Client* mengakses *FTP server* dengan mengirimkan *request FTP* melalui port 21 *load balancer* dan diteruskan oleh *load balancer* menuju *FTP server*. *FTP server* kemudian memproses request tersebut, dan menyambung koneksi dengan *server storage NFS* untuk melakukan proses pengolahan data sesuai dengan isi pesan *request* yang dikirim oleh *client*. *Server storage NFS* melakukan sinkronisasi data apabila terjadi perubahan pada *file-file* yang disimpan. Kemudian *FTP server* mengembalikan respon dari *request* yang dikirim kepada *load balancer* melalui *port* 21. Terakhir *load balancer* meneruskan respon dari *FTP server* menuju alamat *IP client* yang telah mengirim *request*, seperti pada **Gambar 4.4** berikut:



Gambar 4. 4 Topologi *FTP Service*

4.2 Implementasi Sistem

Pada bab implementasi ini dijelaskan langkah-langkah dalam melakukan pembuatan *environment sistem load balancing* dan *cluster server multi-service*. Berdasarkan perancangan yang telah dibahas sebelumnya, *environment sistem load balancing* pada kluster *server multi-service* ini terdiri dari 4 buah *server* dan 1 buah *load balancer*. 3 buah *server* berfungsi sebagai *server-server multi-service* dan 1 buah *server* berfungsi sebagai *storage server*.

4.2.1 Proses Instalasi dan Konfigurasi *Server Multi-Service*

Pada *server multi-service*, digunakan 3 buah *server multi-service* dan 1 buah *server storage* dengan spesifikasi yang sama. Spesifikasi *server-server* tersebut adalah sebagai berikut:

Processor : *Virtual Processor*
Memory : 512 MB
Hardisk : 3.5 GB
OS : *Ubuntu Server 12.04*

Untuk pengalamatan *IP* dari masing-masing *server multi-service* dan *server storage* tersebut, diberikan *IP address* 192.168.186.80 untuk *server storage*, *IP address* 192.168.186.51 untuk *server1*, *IP address* 192.168.186.52 untuk *server2*, dan *IP address* 192.168.186.53 untuk *server 3*. Adapun alamat *gateway* yang digunakan adalah 192.168.186.1.

Gateway merupakan *IP address network interface* yang berfungsi untuk menghubungkan *kluster server multi-service* dan jaringan publik. *Server multi-service* merupakan s*erver* yang berfungsi untuk melayani *request* yang dikirimkan oleh *client*. Setiap *server* tersebut memiliki 2 jenis *service*. *Service* yang berjalan pada *server-server* tersebut adalah *HTTP service dan FTP service*. Untuk *storage server* berfungsi untuk penyimpanan dan sinkronisasi data terhubung dengan ketiga *server multi-service*.

4.2.1.1 *Instalasi dan konfigurasi HTTP Service*

Untuk dapat melakukan pelayanan terhadap *request HTTP* yang dikirimkan oleh *client*, terlabih dahulu dilakukan instalasi dan konfigurasi *HTTP service* pada *server-server multi-service*. Aplikasi yang digunakan untuk dapat melayani *HTTP service* adalah *Apache web server*. Instalasi *Apache web server* dilakukan menggunakan perintah *sudo apt-get install apache2 -y*. Ketika proses instalasi sudah selesai, selanjutnya dilakukan konfigurasi agar *Apache web server* dapat berjalan dengan benar dan mampu melayani *request HTTP*. File konfigurasi yang harus diubah adalah *file 000-default* yang berada pada direktori */etc/apache2/sites-enabled*. Perintah yang digunakan adalah *sudo nano /etc/apache2/sites-enabled/000-default.*

Konfigurasi dilakukan dengan mengubah direktori data dokumen. Direktori data dokumen ini berisikan *file-file* dari sebuah situs yang dijalankan. Sebelumnya terlebih dahulu membuat sebuah direktori sebagai tempat menyimpan file-file situs yang ingin digunakan. Pada kali ini, penulis membuat direktori '*web*' yang terletak pada */var/www/web*. Kemudian pada *file* konfigurasi *000-default*, masukkan alamat direktori yang telah dibuat agar sistem dapat mengetahui letak direktori data dokumen. Konfigurasi *file Apache web server* dapat dilihat pada gambar berikut:

```
 GNU nano 2.2.6    File: /etc/apache2/sites-enabled/000-default

<VirtualHost *:80>
        ServerAdmin webmaster@localhost

        DocumentRoot /var/www/web
        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/web>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
        </Directory>
```

Gambar 4. 5 Konfigurasi *Apache Web Server*

4.2.1.2 *Instalasi dan konfigurasi FTP Service*

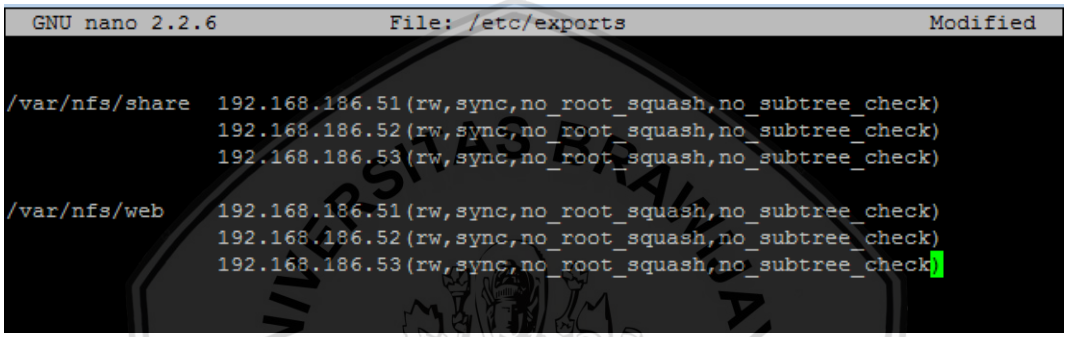
Untuk dapat melakukan pelayanan terhadap *request FTP* yang dikirimkan oleh *client*, terlabih dahulu dilakukan instalasi dan konfigurasi *FTP service* pada *server-server multi-service*. Adapun aplikasi yang digunakan untuk dapat melayani *FTP service* adalah *VsFTPd*. Instalasi *VsFTPd* dilakukan menggunakan perintah *sudo apt-get install VsFTPd -y* . Ketika proses instalasi sudah selesai, selanjutnya dilakukan konfigurasi agar *VsFTPd* dapat berjalan dengan benar dan mampu melayani *request FTP. File* konfigurasi yang harus diubah adalah *file vsFTPd.conf* yang terletak pada direktori /etc/vsFTPd.conf. Perintah yang dimasukkan untuk mengubah file konfigurasi *vsFTP* adalah *sudo nano /etc/vsFTPd.conf*. Berikut adalah konfigurasi file *vsFTPd.conf*:

Gambar 4. 6 Konfigurasi *VsFTPd*

Pada file konfigurasi *vsftopd.conf* perlu dilakukan penyesuaian dengan mengubah parameter *anonymous_enable=NO* agar *FTP* tidak dapat diakses oleh publik. Parameter selanjutnya yang dirubah adalah *pasv_enable=YES* dan *port_enable=YES* untuk mengaktifkan *mode passive* pada layanan *FTP*. Pada masing-masing *server FTP* diberikan *port range* yang berbeda sebagai *port* dimana proses transfer data dilakukan. Untuk *server1 minimum passive port* yang diberikan adalah 10000 dan *maximum passive port* 10250. Pada *server2 minimum passive port* yang diberikan adalah 10251 dan *maximum passive port* 10500. Sedangkan *server3* diberikan *minimum passive port* 10501 dan *maximum passive port* 10700. Dan menambahkan baris kode *pasv_address*=192.168.186.10 untuk untuk mengganti alamat *IP server* menggunakan alamat *eksternal IP* sebagai tanggapan terhadap perintah *PASV* yang dikirim oleh klien.

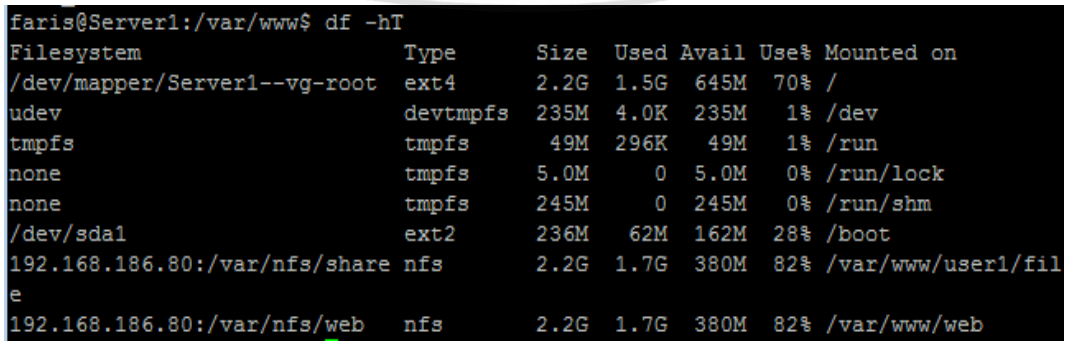
4.2.1.3 *Instalasi dan konfigurasi Storage Server*

Server storage merupakan *server* yang berfungsi untuk menyimpan data dan melakukan sinkronisasi data pada setiap *server-server multi-service* yang terhubung. Pada penelitian ini, digunakan layanan *NFS* untuk melakukan sinkronisasi data. Untuk dapat melakukan proses sinkronisasi data, terlebih dahulu dilakukan instalasi layanan *NFS* dengan menggunakan perintah *sudo apt-get install nfs-kernel-server -y*. Setelah proses instalasi telah selesai, dilakukan konfigurasi pada *file exports* dengan perintah *sudo nano /etc/exports*. Masukkan alamat direktori yang tersambung dengan setiap *server multi-service*. Selain itu dilakukan pengaturan untuk memberikan izin akses kepada setiap *server* dan layanan sinkronisasi pada folder tersebut agar setiap *server* dapat mengakses *file* yang serupa setiap saat.



Gambar 4. 7 Konfigurasi *NFS*

Kemudian pada masing-masing *server multi-service* dilakukan instalasi dan konfigurasi *nfs agent* agar dapat terhubung dan tersinkronisasi dengan *server storage*. Untuk instalasi *nfs agent* dilakukan dengan menggunakan perintah *sudo apt-get install nfs-common –y*. Setelah proses instalasi telah selesai dilakukan konfigurasi untuk menghubungkan direktori pada *server multi-service* dengan direktori pada *server storage*. Kemudian lakukan pengecekan layanan *nfs* pada *server storage* untuk mengetahui apakah setiap *server multi-service* telah tersambung dengan benar. Perintah yang digunakan adalah *df-hT*.



Gambar 4. 8 Layanan *NFS*

4.2.2 Proses Instalasi dan Konfigurasi *Load Balancer*

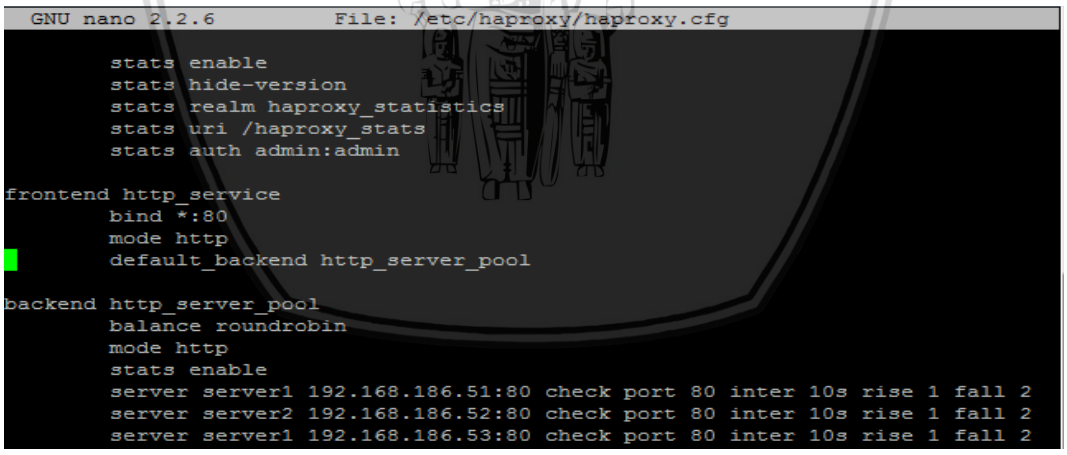
Pada *sistem load balancing*, digunakan 2 buah *server* dengan spesifikasi yang sama. 1 buah *server* untuk *load balancer Haproxy* dan 1 buah untuk *load balancer Zevenet*. Spesifikasi *server-server* tersebut adalah sebagai berikut:

Processor : *Virtual Processor*
Memory : 512 MB
Hardisk : 4.0 GB

Untuk pengalamatan *IP* dari *load balancer* tersebut, diberikan *IP address* 192.168.166.10 untuk *load balancer HAproxy* dan 192.168.166.11 untuk *load balancer Zevenet*. *Load balancer zevenet* memiliki *virtual IP* 192.168.166.10 sebagai alamat *IP* yang diakses oleh *clien*t. Model jaringan yang digunakan kedua *load balancer* ini adalah *Host-Only* yang terhubung dengan jaringan client.

4.2.2.1 *Instalasi dan konfigurasi Haproxy*

Pada *server load balancer 1* sebelumnya telah dilakukan instalasi *Ubuntu Server* sebagai *operating system*. Untuk dapat menggunakan layanan *load balancing*, dilakukan proses instalasi dan konfigurasi *Haproxy*. Perintah yang digunakan adalah *sudo apt-get install haproxy –y*. Setelah *haproxy* sudah terinstal dengan benar, selanjutnya melakukan proses konfigurasi *haproxy*. Proses konfigurasi ini dilakukan dengan mengubah baris pengaturan dari file *haproxy.cfg*. Perintah yang digunakan untuk mengubah konfigurasi *haproxy* adalah *sudo nano /etc/haproxy/haproxy.cfg*.



Gambar 4. 9 Konfigurasi *HAproxy* pada *service HTTP*

Baris perintah *frontend* merupakan inisialisasi *load balancer HTTP_service* yang berfungsi untuk melayani *request HTTP*. Pada *load balancer haproxy* diatur untuk mengikat semua *request* yang masuk pada *port* 80 dan meneruskan menuju *backend server*. Kemudian pada baris perintah *backend server HTTP_server_pool*, diberikan pengaturan untuk melakukan penyeimbangan beban menggunakan algoritme *roud-robin*.

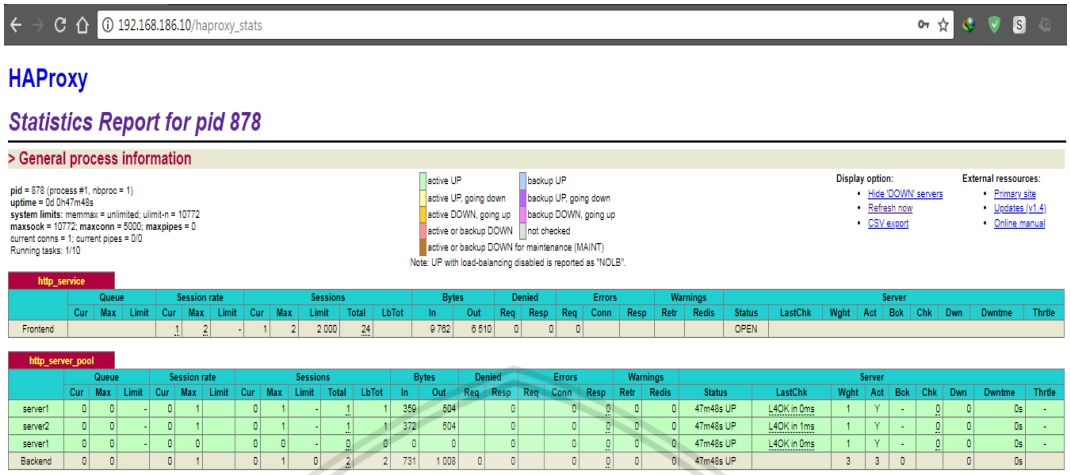
Perintah untuk mengaktifkan statistik dari *server-server multi-service* juga diaktifkan. Kemudian dimasukkan alamat dari setiap *server multi-service* agar *load balancer* dapat meneruskan *request* yang dikirimkan oleh *client*. Alamat *IP* 192.168.186.51 untuk *server 1*, alamat *IP* 192.168.186.52 untuk *server 2* dan alamat *IP* 192.168.186.53 untuk *server 3*.

Untuk service *FTP* dilakukan konfigurasi pada baris *frontend FTP_service* berfungsi sebagai penerima koneksi awal dari *client* dan sebagai kontrol *traffic FTP*. *Frontend* tersebut mengikat setiap *request* yang masuk melalui *port* 21 dan meneruskan kepada *backend FTP_server_pool*. Pada baris *backend* dilakukan inisialisasi masing-masing *server multi-service*, dimana *server1* dengan alamat *IP* 192.168.186.51, *server2* dengan alamat *IP* 192.168.186.52 *dan server3* dengan alamat *IP* 192.168.186.53. Setelah itu merepresentasikan *frontend* dan *backend* untuk setiap *server muti-service*. Pada *server1* mengikat *port* pada *range* 10000 sampai 10250, kemudian *server2* mengikat *port* pada *range* 10251 sampai 10500 dan terakhir *server3* mengikat *port* pada *range* 10501 sampai 10750. Seperti yang tertera pada **Gambar 4.13** dibawah.

```
frontend ftp_service
        bind *:21
        default_backend ftp_server_pool

frontend server1
        bind *:10000-10250
        default_backend server1

frontend server2
        bind *:10251-10500
        default_backend server2

frontend server3
        bind *:10501-10750
        default_backend server3

backend ftp_server_pool
        mode tcp
        balance roundrobin
        server server1 192.168.186.51 check port 21 inter 10s rise 1 fall 2
        server server2 192.168.186.52 check port 21 inter 10s rise 1 fall 2
        server server3 192.168.186.53 check port 21 inter 10s rise 1 fall 2

backend server1
        server server1 192.168.186.51 check port 21 inter 10s rise 1 fall 2

backend server2
        server server2 192.168.186.52 check port 21 inter 10s rise 1 fall 2

backend server3
        server server3 192.168.186.53 check port 21 inter 10s rise 1 fall 2
```

Gambar 4. 10 Konfigurasi *HAproxy* pada *service FTP*

Setelah selesai mengatur konfigurasi *haproxy*, dilakukan pengecekan layanan *haproxy* pada *web-gui haproxy* dengan mengakses alamat 192.168.186.10/haproxy_stats. Seperti yang tertera pada **Gambar 4.14** dibawah:



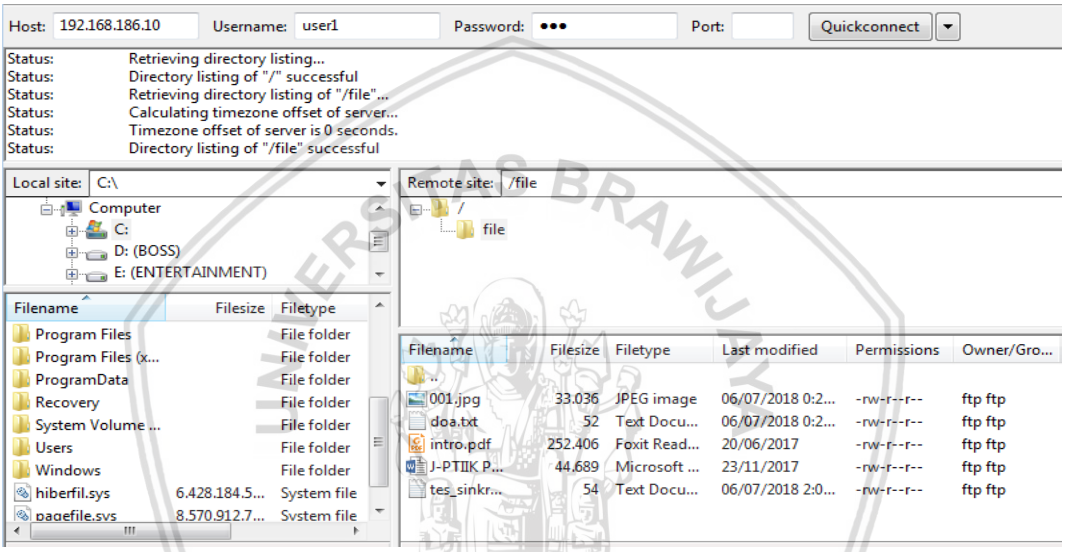
Gambar 4. 11 Statistik *server* pada *HAproxy*

Pada tampilan *web-gui haproxy*, terdapat informasi mengenai *frontend load balancer* dan *server-server backend*. Warna hijau pada *server-server backend* mengindikasikan status aktif. Terdapat informasi mengenai jumlah data yang diterima maupun yang dikirimkan oleh *server-server backend*.

Kemudian melakukan uji akses untuk menentukan apakah fungsi *load balancing* dari *haproxy* sudah dapat berjalan dengan benar atau tidak. Untuk menguji *load balacing haproxy* pada *service HTTP* dilakukan dengan menggunakan *web browser* pada sisi *client*. *Client* mengirim request pada alamat HTTP://192.168.186.10 yang merupakan alamat *IP* dari *haproxy* dan kemudian *haproxy* mendistribusikan *request* tersebut ke setiap *server multi-service* dan mengembalikan *response* dari *server multi-service* menuju *client* seperti yang tertera pada **Gambar 4.15** berikut.

Gambar 4. 12 *HAproxy* pada layanan *HTTP*

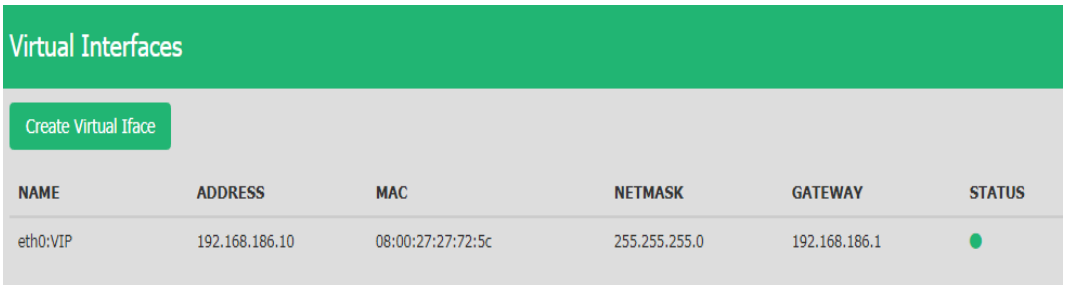
Untuk menguji *load balancing haproxy* pada *service FTP* dilakukan dengan menggunakan *filezilla* pada sisi *client*. *Client* mengirim *request* pada alamat FTP://192.168.186.10 yang merupakan alamat *IP* dari *haproxy* dengan *username* dan *password* pada *port* 21. Kemudian *haproxy* mendistribusikan *request* tersebut ke setiap *server multi-service* dan mengembalikan *response* dari *server multi-service* yang berupa direktori *listing* menuju *client* seperti yang tertera pada **Gambar 4.16** berikut:



Gambar 4. 13 *HAproxy* pada layanan *FTP*

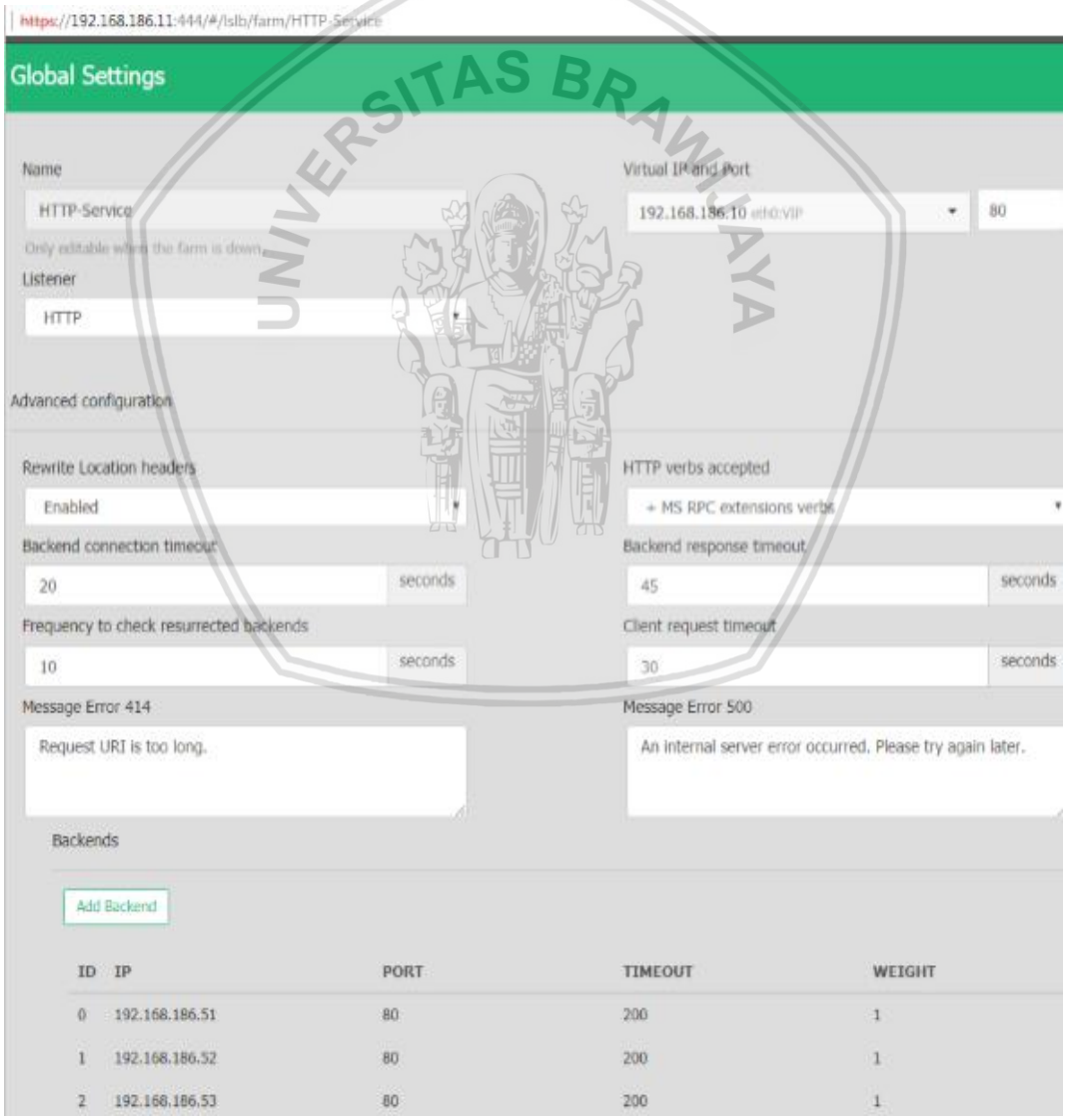
4.2.2.2 *Instalasi dan konfigurasi Zevenet*

Pada *server* 2 *load balancer* dilakukan instalasi *load balancer zevenet*. *Zevenet* merupakan *load balancer* yang berbasis sistem operasi *Debian*. Pada saat instalasi berjalan, penulis memasukkan alamat *IP* 192.168.186.11 mengkonfigurasi alamat *gateway* 192.168.186.1. Setelah proses instalasi telah selesai, kemudian dilakukan proses konfigurasi *load balancer zevenet*. Tidak seperti *Haproxy*, proses konfigurasi *zevenet* dilakukan pada *web-gui zevenet*. *Web-gui zevenet* dapat diakses melalui *web browser* dengan memasukkan alamat HTTPs://192.168.186.11:444.



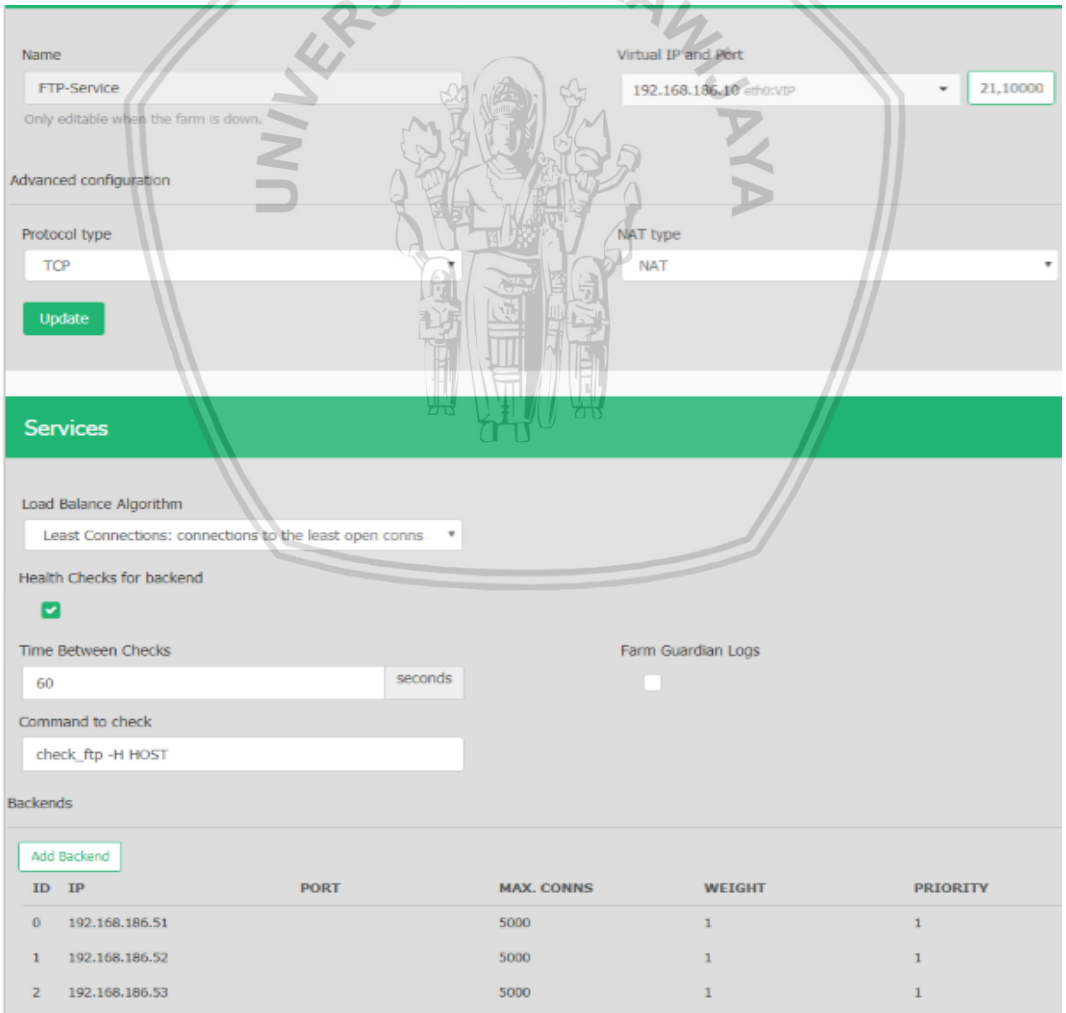
Gambar 4. 14 Membuat *Virtual Interfaces pada Zevenet*

Proses konfigurasi *load balancer zevenet* dilakukan dengan terlebih dahulu membuat sebuah *network interface* baru dengan nama *eth0:VIP* dengan *virtual IP adress* 192.168.186.10, Seperti yang tertera pada **Gambar 4.17** diatas. *Virtual IP* ini merupakan alamat *IP* dari *zevenet* yang dituju oleh *client* ketika mengirim *request*. Selajutnya membuat sebuah *farm HTTP-service* untuk melayani *service HTTP*. Kemudian memasukkan alamat *VIP* yang telah dikonfigurasi beserta *port listening HTTP*, yakni *port* 80. Setelah itu memasukkan alamat *IP* dari setiap *server multi-service*, yakni alamat *IP* 192.168.186.51 untuk *server1*, alamat *IP* 192.168.186.52 untuk *server2* dan alamat *IP* 192.168.186.53 untuk *server3*. Seperti yang tertera pada **Gambar 4.18** berikut:



Gambar 4. 15 Konfigurasi farm *HTTP* pada *Zevenet*

Kemudian pada *farm FTP-service* dilakukan konfigurasi dengan memberikan *VIP* sebagai alamat *IP* yang diakses oleh klien dan *listening port* 21 dan *port range* 10000:10750. *Port range* yang diberikan merupakan akumulasi *port range* yang digunakan setiap *server multi-service* untuk masuk dalam mode *passive FTP*. Protokol yang digunakan merupakan protokol *tcp* untuk proses transfer data menggunakan *passive mode* dan menggunakan *NAT* pada saat meneruskan paket data menuju *server backend*. Algoritme *load balancing* menggunakan *least connections* dan untuk algoritme *round-robin* menggunakan metode *weighted* dengan nilai beban 1 untuk masing-masing server backend. Selanjutnya menambahkan server *backend* beserta jumlah maksimal koneksi yang dapat diterima sebanyak 5000 koneksi. Untuk melakukan pengecekan *backend server* menggunakan baris perintah *check_FTP –H HOST*. Seperti yang tertera pada **Gambar 4.19** berikut:

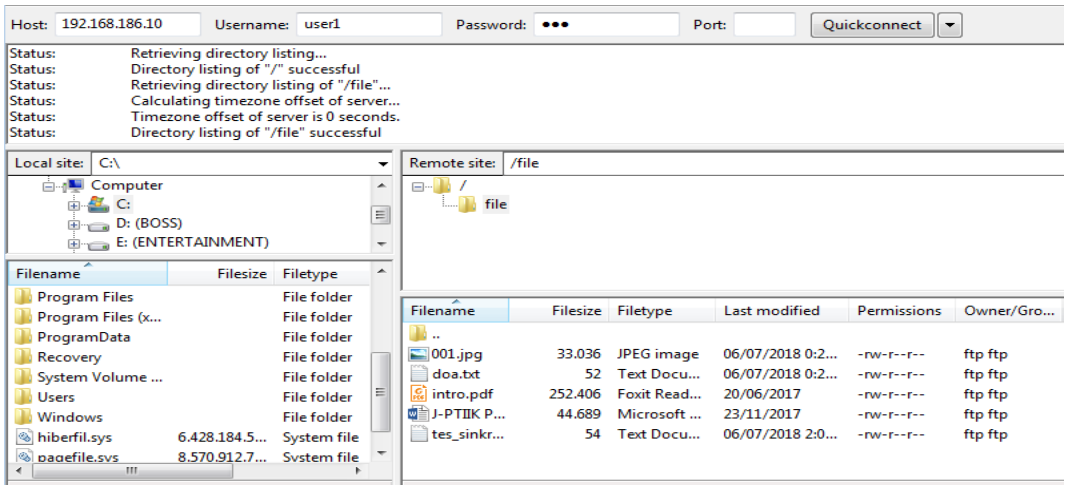


Gambar 4. 16 Konfigurasi Farm *FTP* pada *Zevenet*

Untuk menguji *load balancing zevenet* pada *service HTTP* dilakukan dengan menggunakan *web browser* pada sisi *client*. *Client* mengirim *request* pada alamat HTTP://192.168.166.10 yang merupakan *virtual IP* dari *zevenet* dan kemudian *zevenet* mendistribusikan *request* tersebut ke setiap *server multi-service* dan mengembalikan *response* dari *server multi-service* menuju *client* seperti yang tertera pada **Gambar 4.20** berikut:

Gambar 4. 17 *Zevenet* pada layanan *HTTP*

Kemudian untuk menguji *load balancing zevenet* pada *service FTP* dilakukan dengan menggunakan *filezilla* pada sisi *client*. *Client* mengirim *request* pada alamat FTP://192.168.166.10 yang merupakan *virtual IP* dari *zevenet* dengan memasukkan *username* dan *password* serta pada *port* 21 sebagai tujuan. Kemudian *zevenet* mendistribusikan *request* tersebut ke setiap *server multi-service* dan mengembalikan *response* dari *server multi-service* yang berupa direktori *listing* menuju *client* seperti yang tertera pada **Gambar 4.21** berikut:



Gambar 4. 18 *Zevenet* pada layanan *FTP*

BAB 5 PENGUJIAN DAN ANALISA HASIL

Pada bab ini akan membahas pengujian kinerja *HAproxy* dan *Zevenet* pada kluster *server multi-service*. Kemudian pada akhir bab dilakukan analisa hasil pengujian kinerja *Haproxy* dan *Zevenet* pada kluster *server multi-servic*e untuk menjadi rujukan dalam pengambilan kesimpulan dari *sistem load balancing*.

5.1 Pengujian Fungsional

Tujuan dilakukan pengujian fungsional ini adalah untuk mengukur keberhasilan sistem dalam untuk kebutuhan fungsional sistem. Parameter yang menjadi tolak ukur keberhasilan dalam pengujian fungsional sistem *multi-service load* balancing adalah:

1. *HAproxy* dan *Zevenet* mampu melayani *request HTTP* dan *FTP* yang dikirimkan oleh *client*.
2. *HAproxy* dan *Zevenet* mampu meneruskan *request* yang dikirimkan *client* menuju kluster *server multi-service*.

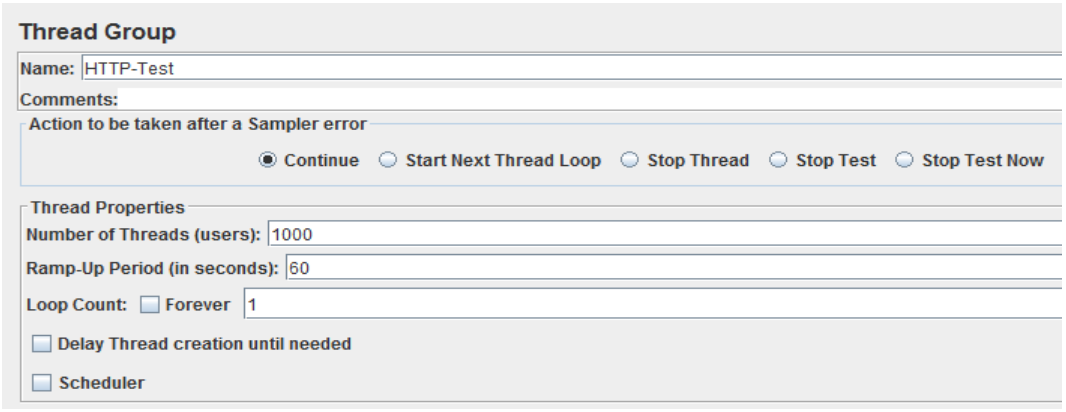
Pengujian fungsional sistem load balancing multi-service HAproxy dan Zevenet dilakukan pada dua tahap, yaitu pengujian fungsional pada layanan HTTP dan pengujian fungsional pada layanan FTP.

5.1.1 Pengujian Fungsional *HTTP Service*

Pengujian fungsional *HTTP service* sistem *load balancer multi-service* dilakukan untuk mengetahui apakah *HAproxy* dan *Zevenet* mampu melayani *request HTTP* yang dikirimkan oleh *client* dan meneruskan *request HTTP* tersebut menuju kluster *server multi-service*.

Langkah pengujian fungsional *HTTP service* sistem *load balancer multi service* adalah:

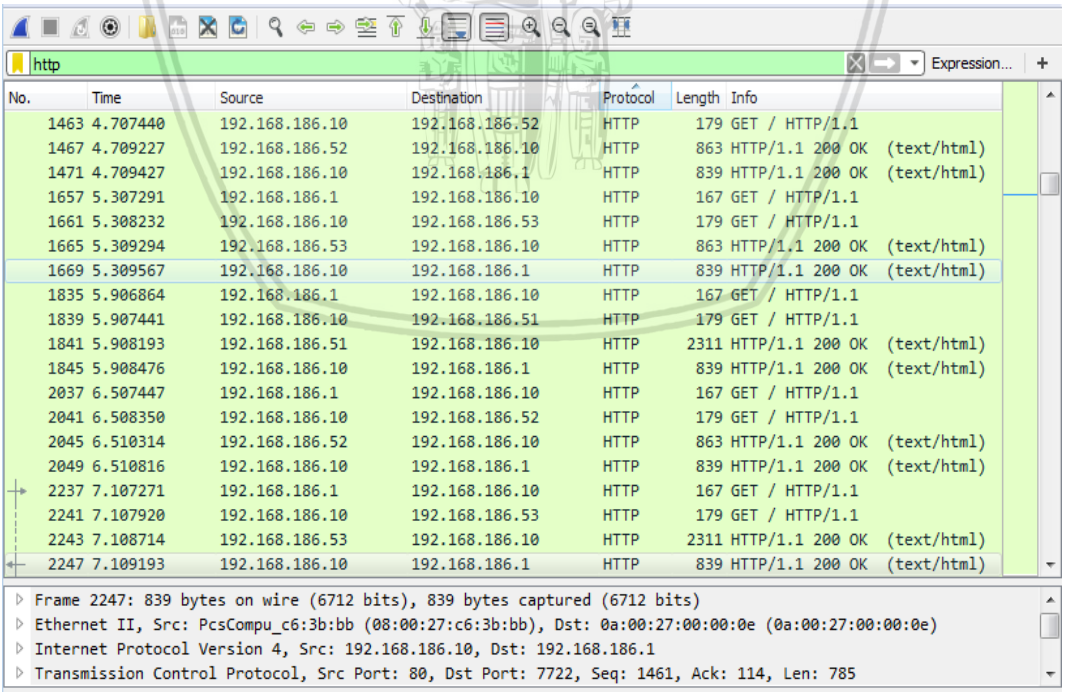
1. *Client* melakukan *request HTTP* pada halaman *index.html* ke *load balancer HAproxy* dan *Zevenet* menggunakan *Apache Jmeter*.
2. *Request* yang dikirimkan sebanyak 1000 koneksi ke alamat *HTTP service load balancer* yaitu 192.168.186.10:80.
3. 1000 *request HTTP* tersebut diteruskan oleh *load balancer* menuju kluster *server backend* yaitu 192.168.186.51, 192.168.186.52 dan 192.168.186.53.
4. Data diambil menggunakan *tcpdump* pada masing-masing *load balancer* dan dianalisa menggunakan *wireshark*.



Gambar 5. 1 Konfigurasi Koneksi *HTTP*

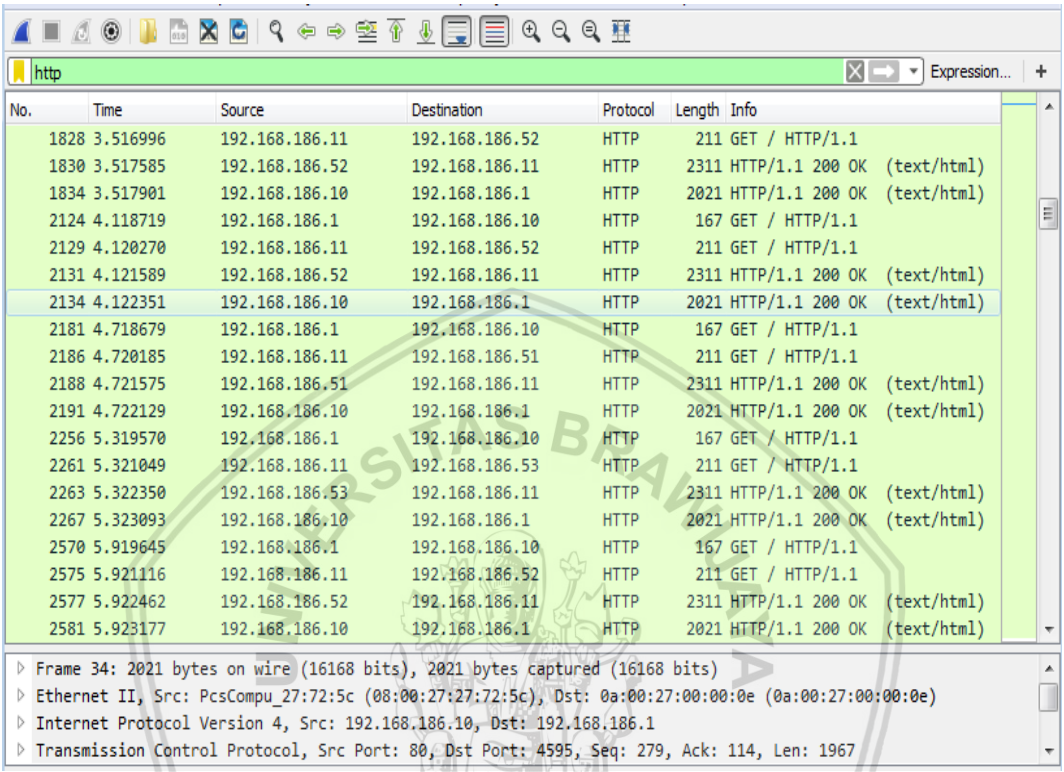
Pada **Gambar 5.1** merupakan konfigurasi *Apache Jmeter* untuk melakukan pengiriman *request HTTP*. *Thread group* merupakan representasi *user* dan diberikan nilai sebanyak 1000 *user* untuk mengirimkan 1000 koneksi. *Ramp-up period* adalah jumlah rentang waktu yang dibutuhkan untuk mengirimkan 1000 *request* dengan nilai 60 detik (1 menit) dan perulangan sebanyak 1 kali.

Pada saat pengujian dilakukan pengambilan data menggunakan *tcpdump* yang diletakkan pada *load balancer HAproxy* dan *Zevenet*. Setelah pengambilan data, dilakukan analisa data dengan menggunakan *wireshark*. Hasil analisa data tersebut menjadi standar pengukuran keberhasilan pengujian fungsional sistem *load balancing multi-service HAproxy* dan *Zevenet* dalam melayani *request HTTP*.



Gambar 5. 2 *Capture Wireshark HTTP HAproxy*

Pada *capture wireshark* diatas terlihat *load balancer Haproxy* dengan alamat ip 192.168.186.10 mampu meneruskan *request HTTP* menuju *server backend* dengan alamat ip 192.168.186.51 untuk *server1*, 192.168.186.52 untuk *server2* dan 192.168.186.53 untuk *server3*. Ketiga *server backend* tersebut mampu mengirimkan halaman *html* yang di*request client*.



Gambar 5. 3 *Capture Wireshark HTTP Zevenet*

Pada **Gambar 5.2** memperlihatkan *capture wireshark* pada layanan *HTTP load balancer Zevenet* dengan alamat ip 192.168.186.11. *Load balancer Zevenet* memiliki *virtual ip* dengan alamat 192.168.186.10 yang merupakan *public address* dari *load balancer Zevenet*. *Zevenet* mampu meneruskan *request HTTP* menuju *server backend* dengan alamat ip 192.168.186.51 untuk server1, 192.168.186.52 untuk server2 dan 192.168.186.53 untuk server3. Ketiga *server backend* tersebut juga mampu mengirimkan halaman *html* yang di*request client*.

Berdasarkan analisa hasil pengujian fungsional sistem *load balancer multi-service HAproxy* dan *Zevenet* pada layanan *HTTP* yang telah dilakukan dapat diketahui masing-masing *load balancer HAproxy* dan *Zevenet* mampu melayani *request HTTP* yang dikirimkan *client* dan meneruskan *request* tersebut menuju ke tiga *server backend.*

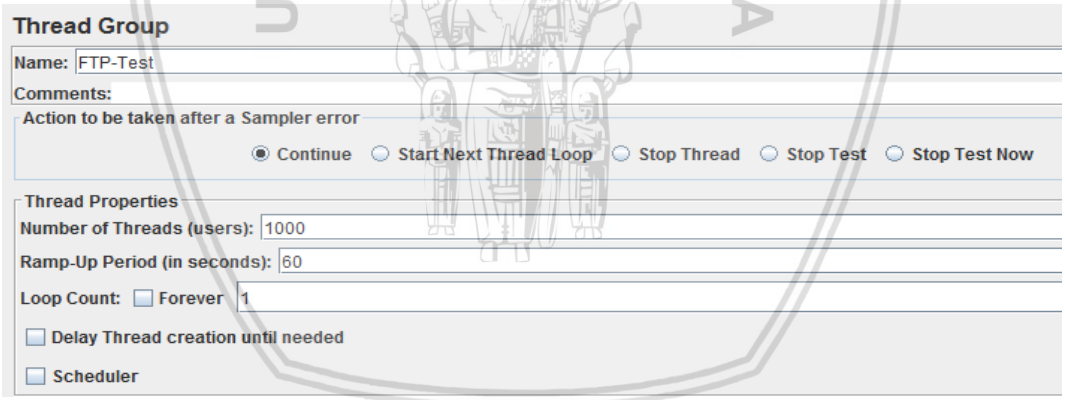
5.1.2 Pengujian Fungsional *FTP Service*

Pengujian fungsional *FTP service* sistem *load balancer multi-service* dilakukan untuk mengetahui apakah *HAproxy* dan *Zevenet* mampu melayani *request FTP* yang dikirimkan oleh *client* dan meneruskan *request FTP* tersebut menuju *kluster server multi-service*.

Langkah pengujian fungsional *FTP service* sistem *load balancer multi service* adalah:

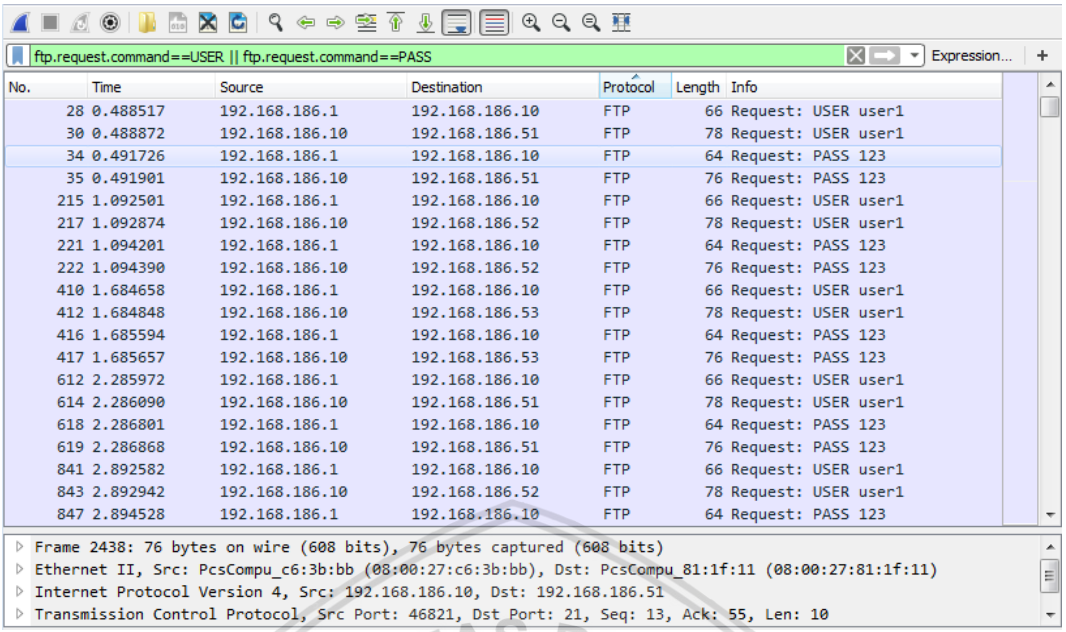
1. *Client* mengirimkan *request FTP* menggunakan *Apache Jmeter* dengan melakukan *login* menggunakan *username "user1"* dan *password "123"* serta mengunduh file *intro.pdf* yang berada pada direktori */file*.
2. *Request* yang dikirim sebanyak 1000 koneksi ke alamat *FTP service load balancer* yaitu 192.168.186.10:21 dan diteruskan menuju kluster *server backend* yaitu 192.168.186.51, 192.168.186.52 dan 192.168.186.53.
3. Data diambil menggunakan *tcpdump* pada *HAproxy* dan *Zevenet* kemudian dianalisa menggunakan *wireshark.*

Pada saat pengujian dilakukan pengambilan data menggunakan *tcpdump* yang diletakkan pada *load balancer HAproxy* dan *Zevenet*. Setelah pengambilan data, dilakukan analisa data dengan menggunakan *wireshark*. Hasil analisa data tersebut menjadi standar pengukuran keberhasilan pengujian fungsional sistem *load balancing multi-service HAproxy* dan *Zevenet* dalam melayani *request FTP*.



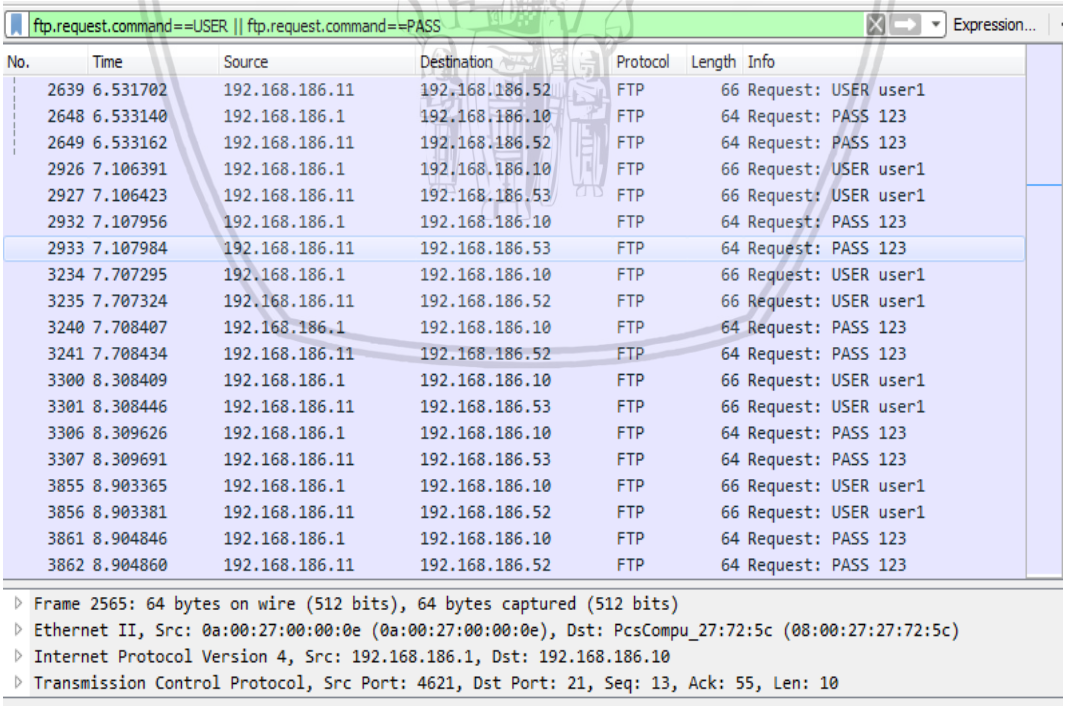
Gambar 5. 4 Konfigurasi Koneksi *FTP*

Pada **Gambar 5.4** merupakan konfigurasi *Apache Jmeter* untuk melakukan pengiriman *request FTP*. *Thread group* merupakan representasi *user* dan diberikan nilai sebanyak 1000 *user* untuk mengirimkan 1000 koneksi. *Ramp-up period* adalah jumlah rentang waktu yang dibutuhkan untuk mengirimkan 1000 *request* dengan nilai 60 detik (1 menit) dan perulangan sebanyak 1 kali.



Gambar 5. 5 *Capture Wireshark FTP HAproxy*

Pada **Gambar 5.1** diatas terlihat *load balancer Haproxy* dengan alamat *ip* 192.168.186.10 mampu meneruskan *request FTP* menuju *server backend* dengan alamat *ip* 192.168.186.51, 192.168.186.52 dan 192.168.186.53. *Client* mampu melakukan *login* dengan menggunakan *username "user1"* dan *password "123"* pada ketiga *server backend*.



Gambar 5. 6 *Capture Wireshark FTP Zevenet*

Pada **Gambar 5.6** memperlihatkan *capture wireshark* pada layanan *FTP load balancer Zevenet* dengan alamat *ip* 192.168.186.11. *Load balancer Zevenet* memiliki *virtual ip* dengan alamat 192.168.186.10 yang merupakan *public address* dari *load balancer Zevenet*. *Zevenet* mampu meneruskan *request FTP* menuju *server backend* dengan alamat *ip* 192.168.186.51, 192.168.186.52 dan 192.168.186.53. Client mampu melakukan *login* dengan menggunakan *username "user1"* dan *password "123"* pada ketiga *server backend*.

Berdasarkan analisa hasil pengujian fungsional sistem *load balancer multi-service HAproxy* dan *Zevenet* pada layanan *FTP* yang telah dilakukan dapat diketahui masing-masing *load balancer HAproxy* dan *Zevenet* mampu melayani *request FTP* yang dikirimkan *client* dan meneruskan *request* tersebut menuju ke tiga server *backend.*

5.2 Pengujian Kinerja

Tujuan dilakukan pengujian kinerja sistem *load balancer multi-service* ini adalah untuk mengukur tingkat performa sistem *load balancer HAproxy* dan *Zevenet* dalam melayani *request HTTP* dan *FTP* yang dikirimkan *client*. Pengujian kinerja dilakukan dengan menjalankan 2 buah skenario, skenario 1 untuk layanan *HTTP*, skenario 2 untuk layanan *FTP*. Skenario pengujian yang digunakan pada pengujian kinerja sistem *load balancing multi service* ini adalah:

1. Skenario 1 *HTTP* dilakukan menguji kinerja *HAproxy* dan *Zevenet* dalam melayani *request HTTP* sebanyak 1000, 2500 dan 5000 koneksi. Skenario ini terdiri dari 4 bagian, yakni pengujian menggunakan *load balancer HAproxy* dengan algoritme *round-robin* dan *least connection* serta pengujian menggunakan *load balancer Zevenet* dengan algoritme *round-robin* dan *least connections*.
2. Skenario 2 *FTP* dilakukan menguji kinerja *HAproxy* dan *Zevenet* dalam melayani *request FTP* sebanyak 100, 2500 dan 5000 koneksi. Skenario ini terdiri dari 4 bagian, yakni pengujian menggunakan *load balancer HAproxy* dengan algoritme *round-robin* dan *least connection* serta pengujian menggunakan *load balancer Zevenet* dengan algoritme *round-robin* dan *least connections*.

Pengambilan data dilakukan menggunakan *Apache Jmeter* pada masing-masing *load balancer HAproxy* dan *Zevenet*. Data kemudian dianalisa berdasarkan nilai parameter *response time* dan *resource utilization*. Hasil analisa data tersebut digunakan sebagai perbandingan kinerja sistem *load balancer HAproxy* dan *Zevenet*. Parameter yang menjadi yang dianalisa pada pengujian kinerja sistem *load balancer multi-service* ini adalah:

1. *Response Time*, Pengukuran parameter ini dilakukan untuk menganalisa waktu tanggap yang dibutuhkan *server* untuk merespon saat paket-paket *request* yang dikirimkan oleh *client*.
2. *Resource Utilization*; Pengukuran parameter ini dilakukan untuk menganalisa seberapa banyak penggunaan sumber daya *CPU* yang digunakan oleh *server* ketika menangani request yang dikirim *client*.

5.2.1 Pengujian Kinerja Skenario 1 *HTTP*

Tujuan dilakukan pengujian kinerja skenario 1 *HTTP* adalah untuk mengetahui performa sistem *load balancing* dalam melakukan distribusi *request HTTP*. Skenario ini terdiri dari 4 bagian, yakni pengujian menggunakan *load balancer HAproxy* dengan algoritme *round-robin* dan *least connection* serta pengujian menggunakan *load balancer Zevenet* dengan algoritme *round-robin* dan *least connections*.

5.2.1.1 Pengujian Skenario 1 *HAproxy* dengan Algoritme *Round Robin*

Pengujian skenario 1 layanan *HTTP* pada *HAproxy* dengan algoritme *round-robin* dilakukan untuk mendapatkan nilai performa *load balancing HAproxy* pada layanan *HTTP* menggunakan algoritme *round-robin* dengan parameter pengujian *response time* dan *resource utilization*.

Langkah pengujian adalah sebagai berikut:

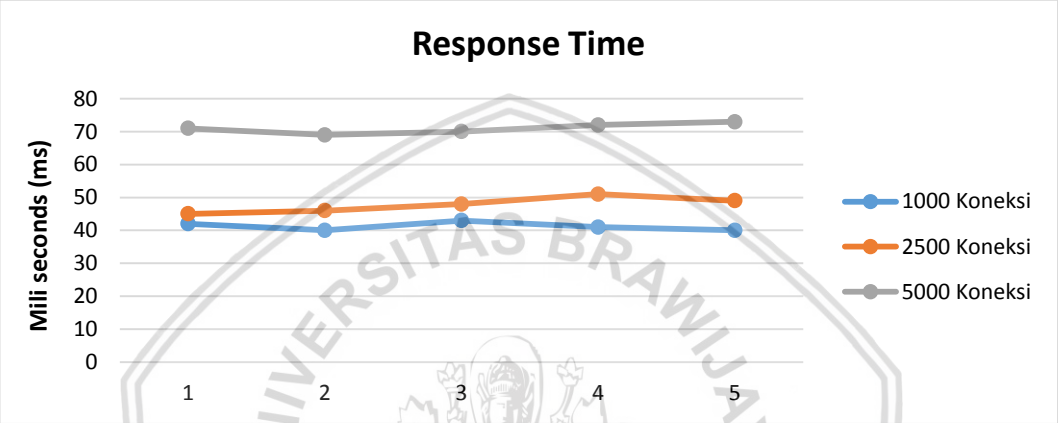
1. *Client* mengirim *request HTTP* pada halaman *index.html* menggunakan *Apache Jmeter* pada *HAproxy*.
2. *Request* yang dikirimkan sebanyak 1000, 2500 dan 5000 koneksi ke alamat *HTTP service load balancer* yaitu 192.168.186.10:80 dan diteruskan menuju *server backend* menggunakan algoritme *round-robin.*
3. Pengujian dilakukan sebanyak 5 kali. Data dengan parameter *response time* dan *resource utilization* diambil menggunakan *plugins Apache Jmeter* dan ditampilkan dalam bentuk tabel.

Hasil pengujian kinerja skenario 1 layanan *HTTP HAproxy* dengan algoritme *round-robin* dengan parameter *response time* adalah:

Tabel 5. 1 *Response Time HTTP HAproxy Round Robin*

***Response Time* (ms)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	42	45	71
2	40	46	69
3	43	48	70
4	41	51	72
5	40	49	73

Nilai parameter *response time* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *HTTP* pada *load balancer HAproxy*. *Request HTTP* yang dikirimkan *client* mengakses halaman *index.html*. Data nilai *response time* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut akan dirubah menjadi grafik dengan mengambil nilai *response time* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *HTTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *HAproxy* pada layanan *HTTP* menggunakan algoritme *round robin* dengan parameter *response time*.



Gambar 5. 7 *Response Time HTTP HAproxy Round-Robin*

Pada **Gambar 5.3** menunjukkan hasil rata-rata pengujian parameter *response time* pada layanan *HTTP load balancer HAproxy* dengan algoritme *round-robin*. Pengujian ini dilakukan sebanyak 5 kali percobaan. Pada percobaan pertama nilai *response time* yang didapat adalah 42ms untuk 1000 koneksi, 45ms untuk 2500 koneksi dan 71ms pada 5000 koneksi. Percobaan kedua didapatkan nilai *response time* yang menurun pada 1000 koneksi sebesar 40ms, 69ms pada 5000 koneksi, sedangkan pada 2500 koneksi nilai *response time* cenderung naik menjadi 46ms. Untuk percobaan ketiga didapatkan nilai 43ms pada 1000 Koneksi, 48ms pada 2500 koneksi dan 70ms untuk 5000 koneksi. Sedangkan percobaan keempat nilai *response time* yang didapat adalah 41ms untuk 1000 koneksi, 51ms untuk 2500 koneksi, dan 72ms untuk 5000 koneksi. Pada percobaan terakhir didapatkan nilai yang cenderung stabil 40ms untuk 1000 koneksi, 49ms untuk 2500 koneksi dan 73ms untuk 5000 koneksi.

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *response time* pada layanan *HTTP load balancer HAproxy* dengan algoritme *round-robin* adalah 43ms untuk 1000 koneksi, 48ms untuk 2500 koneksi, dan 71ms untuk 5000 koneksi. Nilai response time berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian request yang dilakukan load balancer HAproxy pada layanan HTTP dengan

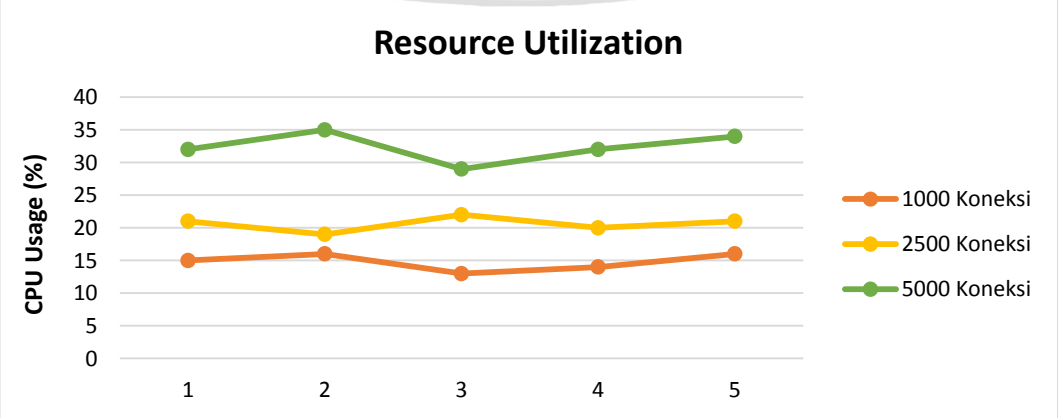
algoritme round robin menunjukkan nilai rata-rata response time yang relatif kecil.

Pengujian juga dilakukan dengan mengambil nilai *resource utilization*. Berikut adalah tabel hasil pengujian kinerja skenario 1 layanan *HTTP HAproxy* dengan algoritme *round-robin* dengan parameter *resource utilization* yang dilakukan sebanyak 5 kali percobaan:

Tabel 5. 2 *Resource Utilization HTTP HAproxy Round Robin*

***Resource Utilization* (CPU%)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	15	21	32
2	16	19	35
3	13	22	29
4	14	20	32
5	16	21	34

Pengujian kinerja skenario 1 layanan *HTTP HAproxy* dengan algoritme *round-robin* dengan parameter *resource utilization* dilakukan sebanyak 5 kali. Nilai parameter *resource utilization* diambil pada saat pengiriman request sebanyak 1000, 2500 dan 5000 koneksi *HTTP* pada *load balancer HAproxy*. *Request HTTP* yang dikirimkan client mengakses halaman *index.html.* Data nilai *response time* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *resource utilization* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *HTTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *HAproxy* pada layanan *HTTP* menggunakan algoritme *round robin* dengan parameter *resource utilization*, seperti yang tertera pada **Gambar 5.4** berikut.



Gambar 5. 8 *Resource Utilization HTTP HAproxy Round-Robin*

Pada percobaan pertama nilai *cpu usage* yang didapat adalah 15% untuk 1000 koneksi, 21% untuk 2500 koneksi dan 32% pada 5000 koneksi. Percobaan kedua didapatkan nilai *cpu usage* yang menurun pada 5000 koneksi sebesar 35%, 19% pada 2500 koneksi, sedangkan pada 1000 koneksi nilai *cpu usage* cenderung naik menjadi 16%. Untuk percobaan ketiga didapatkan nilai 13% pada 1000 Koneksi, 22% pada 2500 koneksi dan 29% untuk 5000 koneksi. Sedangkan percobaan keempat nilai *cpu usage* yang didapat adalah 14% untuk 1000 koneksi, 20% untuk 2500 koneksi, dan 32% untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung naik, 16% untuk 1000 koneksi, 21% untuk 2500 koneksi dan 34% untuk 5000 koneksi.

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *resource utilization* pada layanan *HTTP load balancer HAproxy* dengan algoritme round-robin adalah 16% cpu usage untuk 1000 koneksi, 21% *cpu usage* untuk 2500 koneksi, dan 32% *cpu usage* untuk 5000 koneksi. Nilai *resource utilization* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian request yang dilakukan load balancer *HAproxy* pada layanan *HTTP* dengan algoritme *round robin* menunjukkan nilai rata-rata *resource utilization* yang relatif kecil karena tidak menyebabkan *overload* pada *server backend.*

5.2.1.2 Pengujian Skenario 1 *HAproxy* dengan Algoritme *Least Connection*

Pengujian skenario 1 layanan *HTTP* pada *HAproxy* dengan algoritme *least connection* dilakukan untuk mendapatkan nilai performa *load balancing HAproxy* pada layanan *HTTP* menggunakan algoritme *least connection* dengan parameter pengujian *response time* dan *resource utilization*.

Langkah pengujian adalah sebagai berikut:

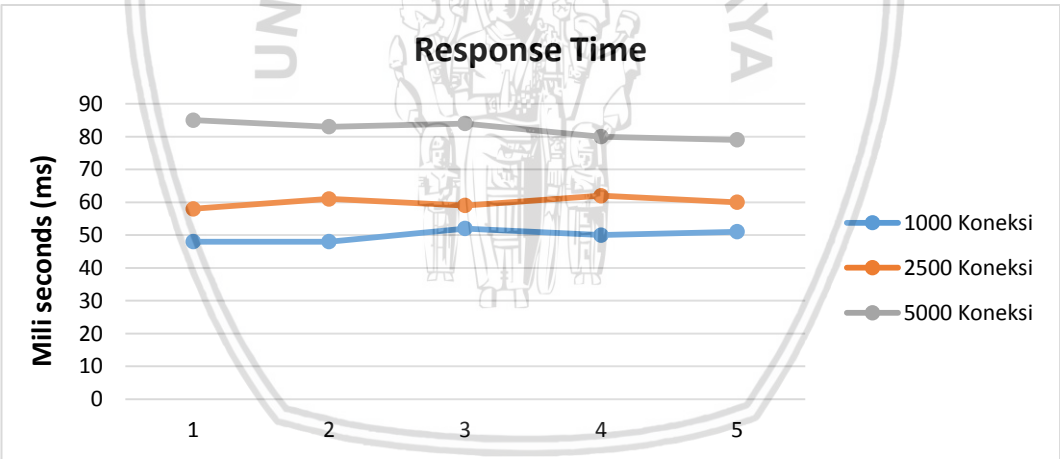
1. *Client* mengirim *request HTTP* pada halaman *index.html* menggunakan *Apache Jmeter* pada *HAproxy.*
2. *Request* yang dikirimkan sebanyak 1000, 2500 dan 5000 koneksi ke alamat *HTTP service load balancer* yaitu 192.168.186.10:80 dan diteruskan menuju *server backend* menggunakan algoritme *least connection.*
3. Pengujian dilakukan sebanyak 5 kali. Data dengan parameter *response time* dan *resource utilization* diambil *menggunakan plugins Apache Jmeter* dan ditampilkan dalam bentuk tabel.

Pengujian dilakukan dengan mengambil nilai *response time*. Berikut adalah tabel hasil pengujian kinerja skenario 1 layanan *HTTP HAproxy* dengan algoritme *least connection* dengan parameter *response time*:

Tabel 5. 3 ***Response Time HTTP HAproxy Least Connection***

***Response Time* (ms)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	48	58	85
2	48	61	83
3	52	59	84
4	50	62	80
5	51	60	79

Nilai parameter *response time* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *HTTP* pada *load balancer HAproxy*. *Request HTTP* yang dikirimkan *client* mengakses halaman *index.html*. Data nilai *response time* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut akan dirubah menjadi grafik dengan mengambil nilai *response time* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi HTTP yang dikirim client. Seperti pada **Gambar 5.11** berikut.



Gambar 5. 9 ***Response Time HTTP HAproxy Least Connection***

Pada percobaan pertama nilai *response time* yang didapat adalah 48ms untuk 1000 koneksi, 58ms untuk 2500 koneksi dan 85ms pada 5000 koneksi. Percobaan kedua didapatkan nilai *response time* yang meningkat pada 1000 koneksi sebesar 48ms, 61ms pada 2500 koneksi, sedangkan pada 5000 koneksi *nilai response time* cenderung turun menjadi 83ms. Untuk percobaan ketiga didapatkan nilai 52ms pada 1000 Koneksi, 59ms pada 2500 koneksi dan 84ms untuk 5000 koneksi. Sedangkan percobaan keempat nilai *response time* yang didapat adalah 50ms untuk 1000 koneksi, 62ms untuk 2500 koneksi, dan 80ms untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung stabil 51ms untuk 1000 koneksi, 60ms untuk 2500 koneksi dan 79ms untuk 5000 koneksi.

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *response time* pada layanan *HTTP load balancer HAproxy* dengan algoritme *least connection* adalah 48ms untuk 1000 koneksi, 60ms untuk 2500 koneksi, dan 83ms untuk 5000 koneksi. Nilai *response time* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian request yang dilakukan *load balancer HAproxy* pada layanan *HTTP* dengan algoritme *least connection* menunjukkan nilai rata-rata *response time* yang lebih kecil dibandingkan dengan algoritme *round-robin* pada jumlah koneksi yang besar. Hal ini disebabkan karena algoritme *least connection* memperhatikan jumlah koneksi pada setiap *server*.

Pengujian juga dilakukan dengan mengambil nilai *resource utilization*. Berikut adalah tabel hasil pengujian kinerja skenario 1 layanan *HTTP HAproxy* dengan algoritme *least connection* dengan parameter *resource utilization* yang dilakukan sebanyak 5 kali percobaan:

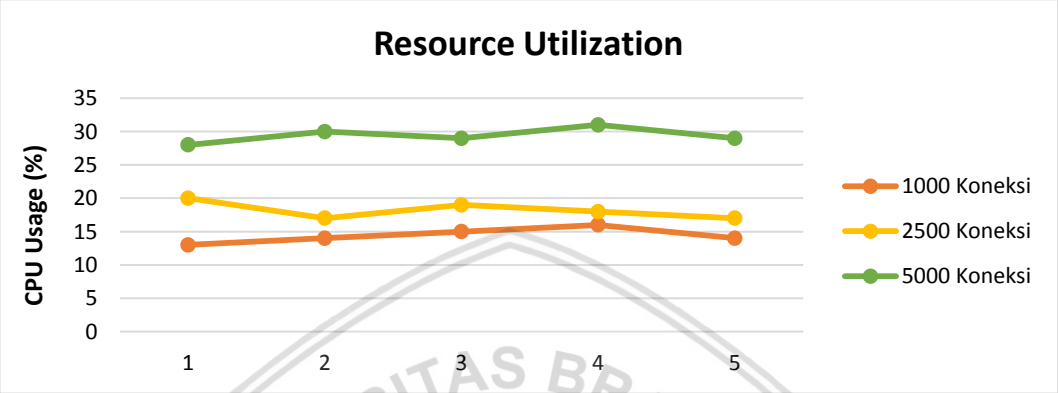
Tabel 5. 4 *Resource Utilization HTTP HAproxy Least Connection*

***Resource Utilization* (CPU%)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	13	20	28
2	14	17	30
3	15	19	29
4	16	18	31
5	14	17	29

Nilai parameter *resource utilization* diambil pada saat pengiriman request sebanyak 1000, 2500 dan 5000 koneksi *HTTP* pada *load balancer HAproxy*. *Request HTTP* yang dikirimkan client mengakses halaman index.html. Data nilai response time diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *resource utilization* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *HTTP* yang dikirimkan *client.* Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *HAproxy* pada layanan *HTTP* menggunakan algoritme *least connection* dengan parameter *resource utilization*. Pengambilan data dilakukan pada salah satu *server backend* dimana nilai *cpu usage* digunakan sebagai standarisasi pengujian.

Pada percobaan pertama nilai *cpu usage* yang didapat adalah 13% untuk 1000 koneksi, 20% untuk 2500 koneksi dan 28% pada 5000 koneksi. Percobaan kedua didapatkan nilai *cpu usage* yang meningkat pada 5000 koneksi sebesar 30%, 14% pada 1000 koneksi, sedangkan pada 2500 koneksi nilai *cpu usage* cenderung stabil dengan 17%. Untuk percobaan ketiga didapatkan nilai 15% pada 1000 Koneksi, 19% pada 2500 koneksi dan 29% untuk 5000 koneksi.

Sedangkan percobaan keempat nilai *cpu usage* yang didapat adalah 16% untuk 1000 koneksi, 18% untuk 2500 koneksi, dan 31% untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung turun, 15% untuk 1000 koneksi, 19% untuk 2500 koneksi dan 29% untuk 5000 koneksi, seperti pada **Gambar 5.12** berikut.



Gambar 5. 10 *Resource Utilization HTTP HAproxy Least Connection*

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *resource utilization* pada layanan *HTTP load balancer HAproxy* dengan algoritme *least connection* adalah 14% *cpu usage* untuk 1000 koneksi, 17% *cpu usage* untuk 2500 koneksi, dan 29% *cpu usage* untuk 5000 koneksi. Nilai resource utilization berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer HAproxy* pada layanan *HTTP* dengan algoritme least connection menunjukkan nilai rata-rata *resource utilization* yang lebih kecil dibandingkan dengan algoritme *round-robin* pada jumlah koneksi yang sedikit dan cenderung sama pada jumlah koneksi yang lebih besar.

5.2.1.3 Pengujian Skenario 1 *Zevenet* dengan Algoritme *Round Robin*

Pengujian skenario 1 layanan *HTTP* pada *Zevenet* dengan algoritme *round-robin* dilakukan untuk mendapatkan nilai performa *load balancing Zevenet* pada layanan *HTTP* menggunakan algoritme *round-robin* dengan parameter pengujian *response time* dan *resource utilization*.

Langkah pengujian adalah sebagai berikut:

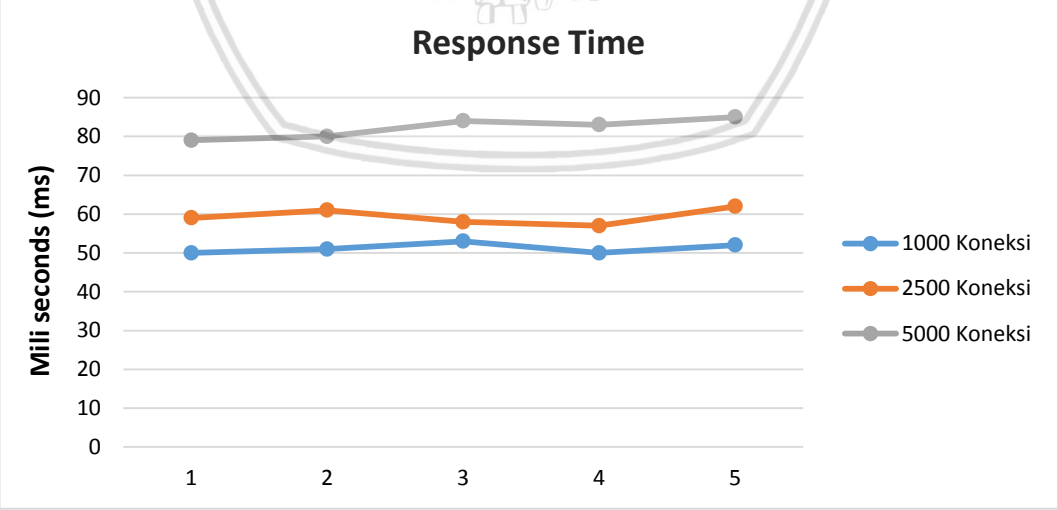
1. *Client* mengirim *request HTTP* pada halaman *index.html* menggunakan *Apache Jmeter* pada *Zevenet*.
2. *Request* yang dikirimkan sebanyak 1000, 2500 dan 5000 koneksi ke alamat *HTTP service load balancer* yaitu 192.168.186.10:80 dan diteruskan menuju *server backend* menggunakan algoritme *round-robin*.
3. Pengujian dilakukan sebanyak 5 kali. Data dengan parameter *response time* dan *resource utilization* diambil menggunakan *plugins Apache Jmeter* dan ditampilkan dalam bentuk tabel.

Pengujian dilakukan dengan mengambil nilai *response time*. Berikut adalah tabel hasil pengujian kinerja skenario 1 layanan *HTTP Zevenet* dengan algoritme *round-robin* dengan parameter *response time*:

Tabel 5. 5 *Response Time HTTP Zevenet Round Robin*

***Response Time* (ms)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	50	59	79
2	51	61	80
3	53	58	84
4	50	57	83
5	52	62	85

Pengujian kinerja skenario 1 layanan *HTTP HAproxy* dengan algoritme *round-robin* dengan parameter *response time* dilakukan sebanyak 5 kali. Nilai parameter *response time* diambil pada saat pengiriman request sebanyak 1000, 2500 dan 5000 koneksi *HTTP* pada *load balancer HAproxy*. *Request HTTP* yang dikirimkan *client* mengakses halaman *index.html.* Data nilai *response time* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut akan dirubah menjadi grafik dengan mengambil nilai *response time* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *HTTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *Zevenet* pada layanan *HTTP* menggunakan algoritme *round robin* dengan parameter *response time*. Pada **Gambar 5.13** menunjukkan hasil rata-rata pengujian parameter *response time* pada layanan *HTTP load balancer Zevenet* dengan algoritme *round-robin*.



Gambar 5. 11 *Response Time HTTP Zevenet Round-Robin*

Pada percobaan pertama nilai *response time* yang didapat adalah 50ms untuk 1000 koneksi, 59ms untuk 2500 koneksi dan 79ms pada 5000 koneksi. Percobaan kedua didapatkan nilai *response time* yang meningkat pada 1000 koneksi sebesar 51ms, 61ms pada 2500 koneksi, sedangkan pada 5000 koneksi *nilai response time* naik menjadi 80ms. Untuk percobaan ketiga didapatkan nilai 53ms pada 1000 Koneksi, 58ms pada 2500 koneksi dan 84ms untuk 5000 koneksi. Sedangkan percobaan keempat nilai *response time* yang didapat adalah 50ms untuk 1000 koneksi, 57ms untuk 2500 koneksi, dan 83ms untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung stabil 52ms untuk 1000 koneksi, 62ms untuk 2500 koneksi dan 85ms untuk 5000 koneksi.

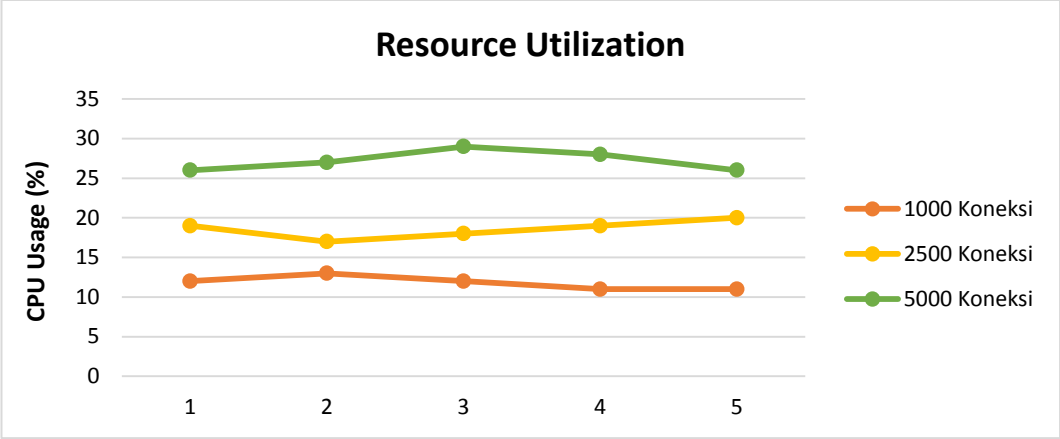
Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *response time* pada layanan *HTTP load balancer Zevenet* dengan algoritme *round-robin* adalah 50ms untuk 1000 koneksi, 59ms untuk 2500 koneksi, dan 83ms untuk 5000 koneksi. Nilai *response time* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer Zevenet* pada layanan *HTTP* dengan algoritme *round robin* menunjukkan nilai rata-rata *response time* yang relatif kecil.

Pengujian juga dilakukan dengan mengambil nilai *resource utilization*. Berikut adalah tabel hasil pengujian kinerja skenario 1 layanan *HTTP HAproxy* dengan algoritme *round-robin* dengan parameter *resource utilization* yang dilakukan sebanyak 5 kali percobaan:

Tabel 5. 6 *Resource Utilization HTTP HAproxy Round Robin*

***Resource Utilization* (CPU%)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	12	19	26
2	13	17	27
3	12	18	29
4	11	19	28
5	11	20	26

Nilai parameter *resource utilization* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *HTTP* pada *load balancer Zevenet*. *Request HTTP* yang dikirimkan *client* mengakses halaman *index.html.* Data nilai *response time* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *resource utilization* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *HTTP* yang dikirimkan *client.* Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *Zevenet* pada layanan *HTTP* menggunakan algoritme *round robin* dengan parameter *resource utilization*.



Gambar 5. 12 *Resource Utilization HTTP Zevenet Round-Robin*

Pengambilan data seperti pada **Gambar 5.8** dilakukan pada salah satu *server backend* dimana nilai *cpu usage* digunakan sebagai standarisasi pengujian. Pada percobaan pertama nilai *cpu usage* yang didapat adalah 12% untuk 1000 koneksi, 19% untuk 2500 koneksi dan 26% pada 5000 koneksi. Percobaan kedua didapatkan nilai *cpu usage* yang meningkat pada 5000 koneksi sebesar 27%, 17% pada 2500 koneksi, sedangkan pada 1000 koneksi nilai *cpu usage* cenderung turun menjadi 13%. Untuk percobaan ketiga didapatkan nilai 12% pada 1000 Koneksi, 18% pada 2500 koneksi dan 29% untuk 5000 koneksi. Sedangkan percobaan keempat nilai *cpu usage* yang didapat adalah 11% untuk 1000 koneksi, 19% untuk 2500 koneksi, dan 28% untuk 5000. Pada percobaan terakhir didapatkan nilai yang stabil, 11% untuk 1000 koneksi, 20% untuk 2500 koneksi dan 26% untuk 5000 koneksi.

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *resource utilization* pada layanan *HTTP load balancer Zevenet* dengan algoritme *round-robin* adalah 11% *cpu usage* untuk 1000 koneksi, 19% *cpu usage* untuk 2500 koneksi, dan 26% *cpu usage* untuk 5000 koneksi. Nilai *resource utilization* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer Zevenet* pada layanan *HTTP* dengan algoritme *round robin* menunjukkan nilai rata-rata *resource utilization* yang relatif kecil karena tidak menyebabkan *overload* pada *server backend.*

5.2.1.4 Pengujian Skenario 1 *Zevenet* dengan Algoritme *Least Connection*

Pengujian skenario 1 layanan *HTTP* pada *Zevenet* dengan algoritme *least connection* dilakukan untuk mendapatkan nilai performa *load balancing Zevenet* pada layanan *HTTP* menggunakan algoritme *least connection* dengan parameter pengujian *response time* dan *resource utilization*.

Langkah pengujian adalah sebagai berikut:

1. *Client* mengirim *request HTTP* pada halaman *index.html* menggunakan *Apache Jmeter* pada *Zevenet*.
2. *Request* yang dikirimkan sebanyak 1000, 2500 dan 5000 koneksi ke alamat *HTTP service load balancer* yaitu 192.168.186.10:80 dan diteruskan menuju *server backend* menggunakan *algoritme least connection.*
3. Pengujian dilakukan sebanyak 5 kali. Data dengan parameter *re*sponse *time* dan *resource utilization* diambil menggunakan *plugins Apache Jmeter* dan ditampilkan dalam bentuk tabel.

Pengujian dilakukan dengan mengambil nilai *response time*. Berikut adalah tabel hasil pengujian kinerja skenario 1 layanan *HTTP Zevenet* dengan algoritme *least connection* dengan parameter *response time*:

Tabel 5. 7 *Response Time HTTP HAproxy Least Connection*

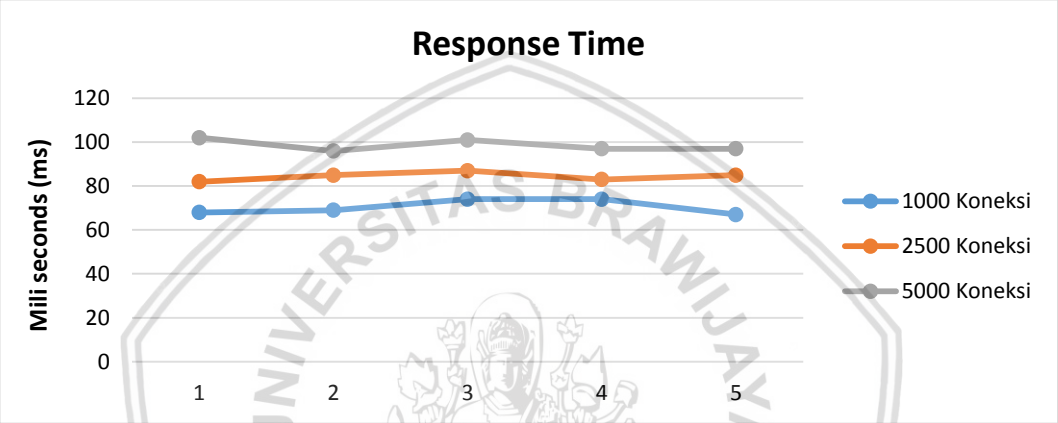
***Response Time* (ms)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	68	82	102
2	69	85	96
3	74	87	101
4	74	83	97
5	67	85	97

Nilai parameter *response time* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *HTTP* pada *load balancer Zevenet*. *Request HTTP* yang dikirimkan *client* mengakses halaman *index.html*. Data nilai *response time* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *response time* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *HTTP* yang dikirimkan *client.* Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *Zevenet* pada layanan *HTTP* menggunakan algoritme *least connection* dengan parameter *response time*. Pengujian ini dilakukan sebanyak 5 kali percobaan.

Pada percobaan pertama nilai *response time* yang didapat adalah 68ms untuk 1000 koneksi, 82ms untuk 2500 koneksi dan 102ms pada 5000 koneksi. Percobaan kedua didapatkan nilai *response time* yang meningkat pada 1000 koneksi sebesar 69ms, 85ms pada 2500 koneksi, sedangkan pada 5000 koneksi nilai *response time* cenderung turun 96ms. Untuk percobaan ketiga didapatkan

nilai 74ms pada 1000 Koneksi, 87ms pada 2500 koneksi dan 101ms untuk 5000 koneksi.

Sedangkan percobaan keempat nilai *response time* yang didapat adalah 74ms untuk 1000 koneksi, 83ms untuk 2500 koneksi, dan 97ms untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung naik 67ms untuk 1000 koneksi, 85ms untuk 2500 koneksi dan 97ms untuk 5000 koneksi. Pada **Gambar 5.15** menunjukkan hasil rata-rata pengujian parameter *response time* pada layanan *HTTP load balancer Zevenet* dengan algoritme *least connection*.



Gambar 5. 13 *Response Time HTTP Zevenet Least Connection*

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *response time* pada layanan *HTTP load balancer Zevenet* dengan algoritme *least connection* adalah 67ms untuk 1000 koneksi, 85ms untuk 2500 koneksi, dan 97ms untuk 5000 koneksi. Nilai *response time* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *reque*st yang dilakukan *load balancer Zevenet* pada layanan *HTTP* dengan algoritme *least connection* menunjukkan nilai rata-rata *response time* yang lebih kecil dibandingkan dengan algoritme *round-robin* pada jumlah koneksi yang besar. Hal ini disebabkan karena algoritme *least connection* memperhatikan jumlah koneksi pada setiap server.

Pengujian juga dilakukan dengan mengambil nilai *resource utilization*. Berikut adalah tabel hasil pengujian kinerja skenario 1 layanan *HTTP Zevenet* dengan algoritme *least connection* dengan parameter *resource utilization* yang dilakukan sebanyak 5 kali percobaan:

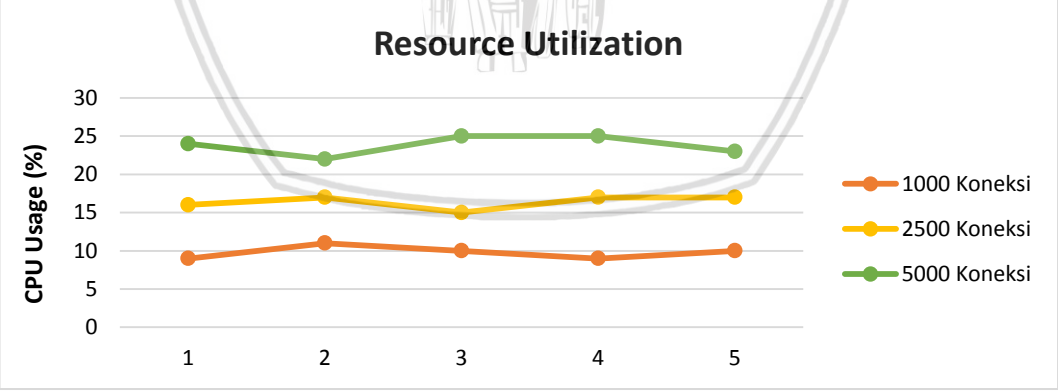
Tabel 5. 8 *Resource Utilization HTTP Zevenet Least Connection*

***Resource Utilization* (CPU%)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	9	16	24
2	11	17	22
3	10	15	25

4	9	17	25
5	10	17	23

Nilai parameter *resource utilization* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *HTTP* pada *load balancer Zevenet*. *Request HTTP* yang dikirimkan *clien*t mengakses halaman *index.html*. Data nilai r*esponse time* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *resource utilization* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *HTT*P yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *Zevenet* pada layanan *HTTP* menggunakan algoritme *least connection* dengan parameter *resource utilization.*

Pengambilan data dilakukan pada salah satu *server backend* dimana nilai *cpu usage* digunakan sebagai standarisasi pengujian. Pada percobaan pertama nilai *cpu usage* yang didapat adalah 9% untuk 1000 koneksi, 16% untuk 2500 koneksi dan 24% pada 5000 koneksi. Percobaan kedua didapatkan nilai *cpu usage* yang turun pada 5000 koneksi sebesar 22%, 11% pada 1000 koneksi, sedangkan pada 2500 koneksi nilai *cpu usage* stabil dengan 17%. Untuk percobaan ketiga didapatkan nilai 10% pada 1000 Koneksi, 15% pada 2500 koneksi dan 25% untuk 5000 koneksi. Sedangkan percobaan keempat nilai *cpu usage* yang didapat adalah 9% untuk 1000 koneksi, 17% untuk 2500 koneksi, dan 25% untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung naik, 10% untuk 1000 koneksi, 17% untuk 2500 koneksi dan 23% untuk 5000 koneksi, seperti pada **Gambar 5.12** berikut.



Gambar 5. 14 *Resource Utilization HTTP Zevenet Least Connection*

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *resource utilization* pada layanan *HTTP load balancer Zevenet* dengan algoritme *least connection* adalah 10% *cpu usage* untuk 1000 koneksi, 17% *cpu usage* untuk 2500 koneksi, dan 25% *cpu usage* untuk 5000 koneksi. Nilai *resource utilization* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan load balancer *Zevenet* pada layanan *HTTP* dengan algoritme *least connection* menunjukkan nilai rata-

rata *resource utilization* yang lebih kecil dibandingkan dengan algoritme *round-robin* pada jumlah koneksi sedikit maupun banyak.

Pada skenario pengujian layanan *HTTP* menggunakan *load balancer HAproxy* dan *Zevenet* diketahui bahwa, *HAproxy* dengan algoritme *round robin* menunjukkan hasil yang lebih baik pada parameter *response time* dengan rata-rata nilai 43ms, 48ms dan 71ms serta pada algoritme *least connection* rata-rata nilai 48ms, 60ms dan 83ms. Untuk parameter *resource utilization, Zevenet* unggul dengan nilai rata-rata *CPU usage* nya 11%, 19%, dan 26% untuk algoritme *round robin*. Sedangkan parameter *resource utilization* dengan algoritme *least connections Zevene*t unggul dengan nilai rata-rata 10%, 17% dan 25%.

5.2.2 Pengujian Kinerja Skenario 2 *FTP*

Tujuan dilakukan pengujian kinerja skenario 2 *FTP* adalah untuk mengetahui performa sistem *load balancing* dalam melakukan distribusi *request FTP*. Skenario ini terdiri dari 4 bagian, yakni pengujian menggunakan *load balancer HAproxy* dengan algoritme *round-robin* dan *least connection* serta pengujian menggunakan *load balancer Zevenet* dengan algoritme *round-robin* dan *least connections*.

5.2.2.1 Pengujian Skenario 2 *HAproxy* dengan Algoritme *Round-Robin*

Pengujian skenario 2 layanan *FTP* pada *HAproxy* dengan algoritme *round-robin* dilakukan untuk mendapatkan nilai performa *load balancing HAproxy* pada layanan *FTP* menggunakan algoritme *round-robin* dengan parameter pengujian *response time* dan *resource utilization*.

Langkah pengujian adalah sebagai berikut:

1. *Client* mengirim *request FTP* dengan mengunduh *file intro.pdf* pada direktori */file* menggunakan *Apache Jmeter* pada *HAproxy.*
2. *Re*quest yang dikirimkan sebanyak 1000, 2500 dan 5000 koneksi ke alamat *FTP service load balancer* yaitu 192.168.186.10:21 dan diteruskan menuju *server backend* menggunakan algoritme *round-robin*.
3. Pengujian dilakukan sebanyak 5 kali. Data dengan parameter *response time* dan *resource utilization* diambil menggunakan *plugins Apache Jmeter* dan ditampilkan dalam bentuk tabel.

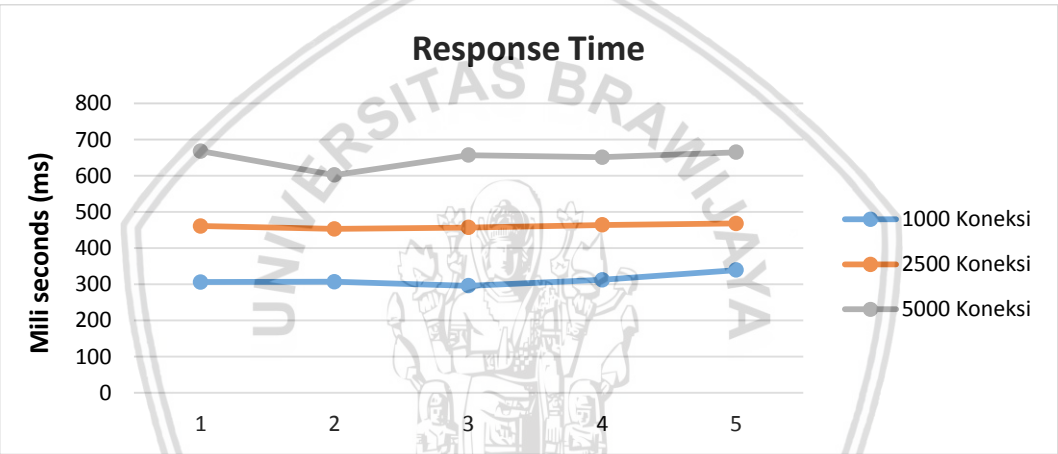
Berikut adalah tabel hasil pengujian kinerja skenario 2 layanan *FTP HAproxy* dengan algoritme *round-robin* dengan parameter *response time*:

Tabel 5. 9 *Response Time FTP HAproxy Round Robin*

***Response Time* (ms)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	306	461	668
2	307	453	602

3	296	457	657
4	312	464	651
5	339	468	665

Nilai parameter *response time* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *FTP* pada *load balancer HAproxy*. *Request FTP* yang dikirimkan *client* mengunduh *file intro.pdf*. Data nilai *response time* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *response time* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *FTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *HAproxy* pada layanan *FTP* menggunakan algoritme *round robin* dengan parameter *response time*.



Gambar 5. 15 *Response Time FTP HAproxy Round-Robin*

Pada percobaan pertama nilai *response time* yang didapat adalah 306ms untuk 1000 koneksi, 461ms untuk 2500 koneksi dan 668ms pada 5000 koneksi. Percobaan kedua didapatkan nilai *response time* yang naik pada 1000 koneksi sebesar 307ms, 453ms pada 2500 koneksi, sedangkan pada 1000 koneksi nilai *response time* cenderung turun menjadi 602ms. Untuk percobaan ketiga didapatkan nilai 296ms pada 1000 Koneksi, 457ms pada 2500 koneksi dan 657ms untuk 5000 koneksi. Sedangkan percobaan keempat nilai *response time* yang didapat adalah 312ms untuk 1000 koneksi, 464ms untuk 2500 koneksi, dan 651ms untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung stabil 339ms untuk 1000 koneksi, 468ms untuk 2500 koneksi dan 665ms untuk 5000 koneksi, seperti pada **Gambar 5.19** diatas.

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *response time* pada layanan *FTP load balancer HAproxy* dengan algoritme *round-robin* adalah 309ms untuk 1000 koneksi, 463ms untuk 2500 koneksi, dan 654ms untuk 5000 koneksi. Nilai *response time* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer HAproxy* pada layanan *FTP* dengan

algoritme *round robin* menunjukkan nilai rata-rata *response time* yang relatif kecil.

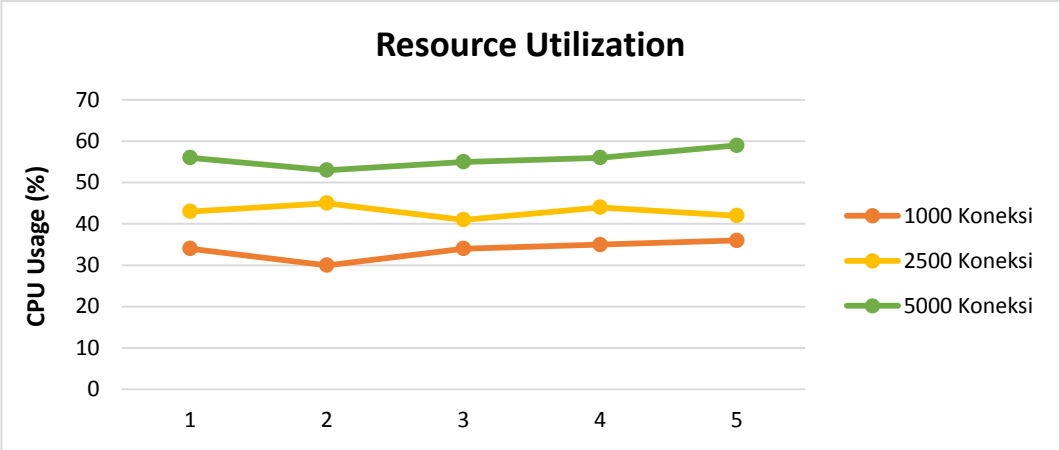
Pengujian juga dilakukan dengan mengambil nilai *resource utilization*. Berikut adalah tabel hasil pengujian kinerja skenario 1 layanan *FTP HAproxy* dengan algoritme *round-robin* dengan parameter *resource utilization* yang dilakukan sebanyak 5 kali percobaan:

Tabel 5. 10 *Resource Utilization FTP HAproxy Round Robin*

***Resource Utilization* (CPU%)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	34	43	56
2	30	45	53
3	34	41	55
4	35	44	56
5	36	42	59

Pengujian kinerja skenario 1 layanan *FTP HAproxy* dengan algoritme *round-robin* dengan parameter *resource utilization* dilakukan sebanyak 5 kali. Nilai parameter *resource utilization* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *FTP* pada load balancer *HAproxy*. *Request FTP* yang dikirimkan *client* mengunduh *file intro.pdf.* Data nilai *cpu usage* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut akan dirubah menjadi grafik dengan mengambil nilai *resource utilization* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *FTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *HAproxy* pada layanan *FTP* menggunakan algoritme *round robin* dengan parameter *resource utilization*.

Pada **Gambar 5.20** menunjukkan hasil rata-rata pengujian parameter *resource utilization* pada layanan *FTP load balancer HAproxy* dengan algoritme *round-robin*. Pengujian ini dilakukan sebanyak 5 kali percobaan. Pengambilan data dilakukan pada salah satu *server backend* dimana nilai *cpu usage* digunakan sebagai standarisasi pengujian. Pada percobaan pertama nilai *cpu usage* yang didapat adalah 34% untuk 1000 koneksi, 43% untuk 2500 koneksi dan 56% pada 5000 koneksi. Percobaan kedua didapatkan nilai *cpu usage* yang naik pada 1000 koneksi sebesar 30%, 45% pada 2500 koneksi, sedangkan pada 5000 koneksi nilai *cpu usage* turun menjadi 53%. Untuk percobaan ketiga didapatkan nilai 34% pada 1000 Koneksi, 41% pada 2500 koneksi dan 55% untuk 5000 koneksi. Sedangkan percobaan keempat nilai *cpu usage* yang didapat adalah 35% untuk 1000 koneksi, 44% untuk 2500 koneksi, dan 56% untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung naik, 36% untuk 1000 koneksi, 42% untuk 2500 koneksi dan 59% untuk 5000 koneksi.



Gambar 5. 16 *Resource Utilization FTP HAproxy Round-Robin*

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *resource utilization* pada layanan *FTP load balancer HAproxy* dengan algoritme *round-robin* adalah 34% *cpu usage* untuk 1000 koneksi, 43% *cpu usage* untuk 2500 koneksi, dan 56% *cpu usage* untuk 5000 koneksi. Nilai *resource utilization* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer HAproxy* pada layanan *FTP* dengan algoritme *round robin* menunjukkan nilai rata-rata *resource utilization* yang relatif kecil karena tidak menyebabkan *overload* pada *server backend.*

5.2.2.2 Pengujian Skenario 2 *HAproxy* dengan Algoritme *Least Connection*

Pengujian skenario 2 layanan *FTP* pada *HAproxy* dengan algoritme *least connection* dilakukan untuk mendapatkan nilai performa *load balancing HAproxy* pada layanan *FTP* menggunakan algoritme *least connection* dengan parameter pengujian *response time* dan *resource utilization*.

Langkah pengujian adalah sebagai berikut:

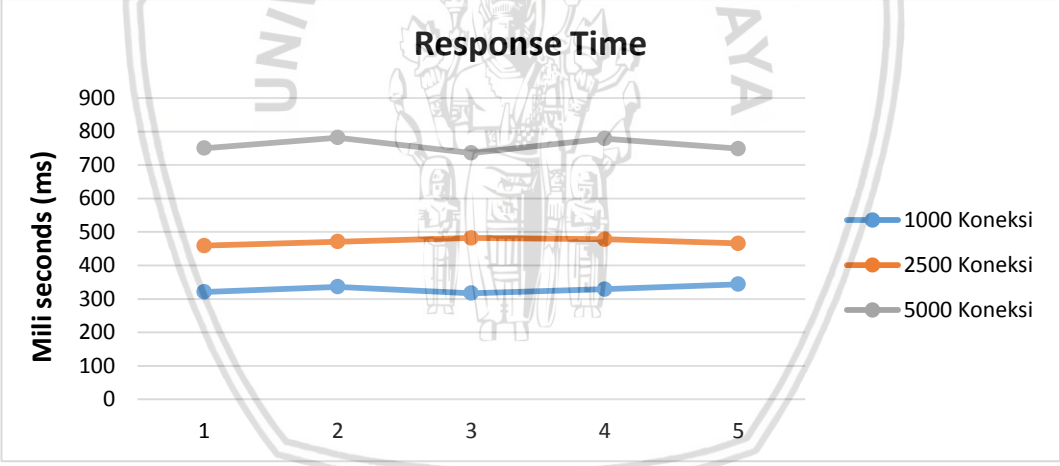
1. *Client* mengirim *request FTP* dengan mengunduh *file intro.pdf* yang berada di direktori */file* menggunakan *Apache Jmeter* pada *HAproxy*.
2. *Reques*t yang dikirimkan sebanyak 1000, 2500 dan 5000 koneksi ke alamat *FTP service load balancer* yaitu 192.168.186.10:21 dan diteruskan menuju *server backend* menggunakan algoritme *least connection*.
3. Pengujian dilakukan sebanyak 5 kali. Data dengan parameter *response time* dan *resource utilization* diambil menggunakan *plugins Apache Jmeter* dan ditampilkan dalam bentuk tabel.

Pengujian dilakukan dengan mengambil nilai response time. Berikut adalah tabel hasil pengujian kinerja skenario 2 layanan *FTP HAproxy* dengan algoritme *least connection* dengan parameter *response time*:

Tabel 5. 11 *Response Time FTP HAproxy Least Connection*

***Response Time* (ms)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	321	459	750
2	336	471	782
3	317	482	736
4	329	478	779
5	344	466	749

Nilai parameter *response time* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *FTP* pada *load balancer HAproxy*. Request *FTP* yang dikirimkan *client* mengunduh file *intro.pdf*. Data nilai *response time* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *response time* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *FTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *HAproxy* pada layanan *FTP* menggunakan algoritme *least connection* dengan parameter *response time*.



Gambar 5. 17 *Response Time FTP HAproxy Algoritme Least Connection*

Pada **Gambar 5.21** menunjukkan hasil rata-rata pengujian parameter *response time* pada layanan *HTTP load balancer HAproxy* dengan algoritme *least connection*. Pengujian ini dilakukan sebanyak 5 kali percobaan. Pada percobaan pertama nilai *response time* yang didapat adalah 321ms untuk 1000 koneksi, 459ms untuk 2500 koneksi dan 720ms pada 5000 koneksi. Percobaan kedua didapatkan nilai *response time* yang naik pada 1000 koneksi sebesar 336ms, 782ms pada 5000 koneksi, sedangkan pada 2500 koneksi nilai *response time* cenderung naik menjadi 471ms.

Untuk percobaan ketiga didapatkan nilai 317ms pada 1000 Koneksi, 482ms pada 2500 koneksi dan 736ms untuk 5000 koneksi. Sedangkan percobaan keempat nilai *response time* yang didapat adalah 329ms untuk 1000 koneksi, 478ms untuk 2500 koneksi, dan 779ms untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung naik 344ms untuk 1000 koneksi, 466ms untuk 2500 koneksi dan 749ms untuk 5000 koneksi.

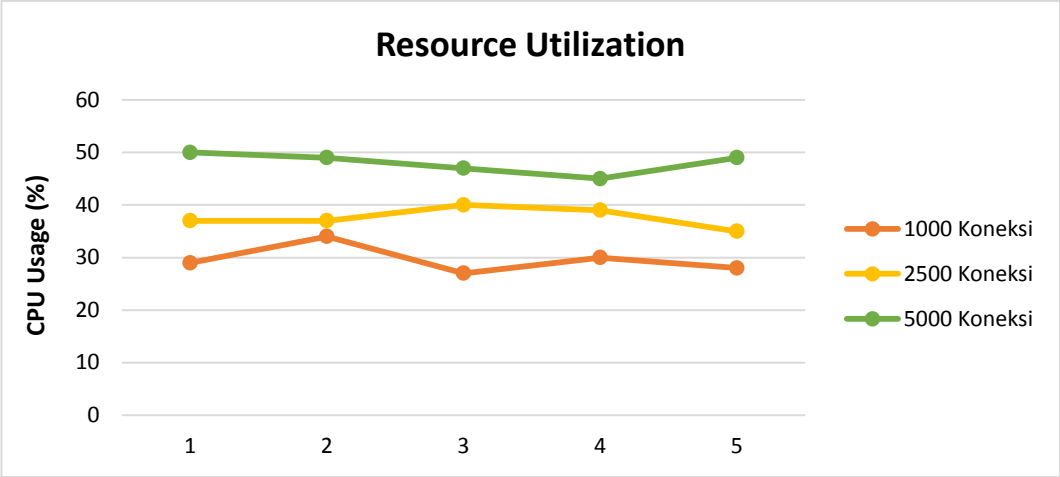
Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *response time* pada layanan *FTP load balancer HAproxy* dengan algoritme *least connection* adalah 328ms untuk 1000 koneksi, 474ms untuk 2500 koneksi, dan 752ms untuk 5000 koneksi. Nilai *response time* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer HAproxy* pada layanan *FTP* dengan algoritme *least connection* menunjukkan nilai rata-rata *response time* yang lebih kecil dibandingkan dengan algoritme *round-robin* pada jumlah koneksi besar maupun kecil. Hal ini disebabkan karena algoritme least connection memperhatikan jumlah koneksi pada setiap *server* dan memilih *server backend* dengan jumlah koneksi paling rendah.

Pengujian juga dilakukan dengan mengambil nilai *resource utilization*. Berikut adalah tabel hasil pengujian kinerja skenario 2 layanan *FTP HAproxy* dengan algoritme *least connection* dengan parameter *resource utilization* yang dilakukan sebanyak 5 kali percobaan:

Tabel 5. 12 *Resource Utilization FTP HAproxy Least Connection*

***Resource Utilization* (CPU%)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	29	37	50
2	34	37	49
3	27	40	47
4	30	39	45
5	28	35	49

Pengujian kinerja skenario 2 layanan *FTP HAproxy* dengan algoritme *least connection* dengan parameter *resource utilization* dilakukan sebanyak 5 kali. Nilai parameter *resource utilization* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *FTP* pada *load balancer HAproxy*. *Request FTP* yang dikirimkan *client* mengunduh file *intro.pdf*. Data nilai *least connections* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *resource utilization* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *FTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *HAproxy* pada layanan *FTP* menggunakan algoritme *least connection* dengan parameter *resource utilization*.



Gambar 5. 18 *Resource Utilization FTP HAproxy Least Connection*

Pada **Gambar 5.22** menunjukkan hasil rata-rata pengujian parameter *resource utilization* pada layanan *FTP load balancer HAproxy* dengan algoritme *least connection*. Pengujian ini dilakukan sebanyak 5 kali percobaan. Pengambilan data dilakukan pada salah satu *server backend* dimana nilai *cpu usage* digunakan sebagai standarisasi pengujian. Pada percobaan pertama nilai *cpu usage* yang didapat adalah 29% untuk 1000 koneksi, 37% untuk 2500 koneksi dan 50% pada 5000 koneksi. Percobaan kedua didapatkan nilai *cpu usage* yang turun pada 1000 koneksi sebesar 24%, 49% pada 5000 koneksi, sedangkan pada 2500 koneksi nilai *cpu usage* stabil pada 37%. Untuk percobaan ketiga didapatkan nilai 27% pada 1000 Koneksi, 40% pada 2500 koneksi dan 57% untuk 5000 koneksi. Sedangkan percobaan keempat nilai *cpu usage* yang didapat adalah 30% untuk 1000 koneksi, 39% untuk 2500 koneksi, dan 45% untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung turun, 28% untuk 1000 koneksi, 35% untuk 2500 koneksi dan 49% untuk 5000 koneksi.

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *resource utilization* pada layanan *FTP load balancer HAproxy* dengan algoritme *round-robin* adalah 28% *cpu usage* untuk 1000 koneksi, 37% *cpu usage* untuk 2500 koneksi, dan 49% *cpu usage* untuk 5000 koneksi. Nilai *resource utilization* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer HAproxy* pada layanan *FTP* dengan algoritme *least connection* menunjukkan nilai rata-rata *resource utilization* yang lebih besar dibandingkan dengan algoritme *round-robin* pada jumlah koneksi besar dan cenderung sama pada jumlah koneksi yang sedikit.

5.2.2.3 Pengujian Skenario 2 *Zevenet* dengan Algoritme *Round-Robin*

Pengujian skenario 2 layanan *FTP* pada *Zevenet* dengan algoritme *round-robin* dilakukan untuk mendapatkan nilai performa *load balancing Zevenet* pada layanan *FTP* menggunakan algoritme *round-robin* dengan parameter pengujian *response time* dan *resource utilization*.

Langkah pengujian adalah sebagai berikut:

1. *Client* mengirim *request FTP* dengan mengunduh *file intro.pdf* pada direktori */file* menggunakan *Apache Jmeter* pada *Zevenet*.
2. *Request* yang dikirimkan sebanyak 1000, 2500 dan 5000 koneksi ke alamat *FTP service load balancer* yaitu 192.168.186.10:80 dan diteruskan menuju *server backend* menggunakan algoritme *round-robin*.
3. Pengujian dilakukan sebanyak 5 kali. Data dengan parameter *response time* dan *resource utilization* diambil menggunakan *plugins Apache Jmeter* dan ditampilkan dalam bentuk tabel.

Berikut adalah tabel hasil pengujian kinerja skenario 2 layanan *FTP Zevenet* dengan algoritme *round-robin* dengan parameter *response time*:

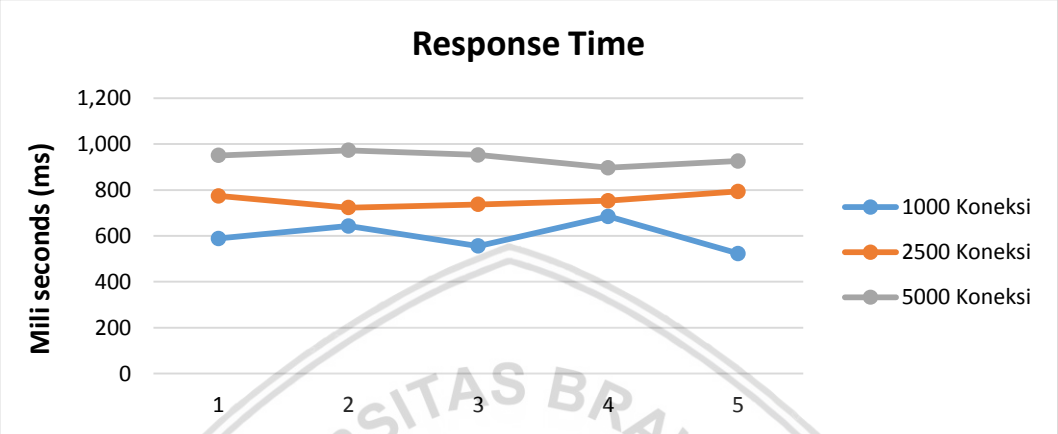
Tabel 5. 13 *Response Time FTP Zevenet Round Robin*

***Response Time* (ms)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	589	774	951
2	643	723	973
3	556	737	953
4	685	753	897
5	523	794	926

Nilai parameter *response time* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *FTP* pada *load balancer Zevenet*. *Request FTP* yang dikirimkan *client* mengunduh *file intro.pdf*. Data nilai *response time* diatas diambil menggunakan plugins *Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *response time* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *FTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *Zevenet* pada layanan *FTP* menggunakan algoritme *round robin* dengan parameter *response time*.

Pada percobaan pertama nilai *response time* yang didapat adalah 589ms untuk 1000 koneksi, 774ms untuk 2500 koneksi dan 951ms pada 5000 koneksi. Percobaan kedua didapatkan nilai *response time* yang naik pada 1000 koneksi sebesar 643ms, 723ms pada 2500 koneksi, sedangkan pada 1000 koneksi nilai *response time* naik menjadi 973ms.

Untuk percobaan ketiga didapatkan nilai 556ms pada 1000 Koneksi, 737ms pada 2500 koneksi dan 953ms untuk 5000 koneksi. Sedangkan percobaan keempat nilai *response time* yang didapat adalah 685ms untuk 1000 koneksi, 753ms untuk 2500 koneksi, dan 897ms untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung naik 523ms untuk 1000 koneksi, 794ms untuk 2500 koneksi dan 926ms untuk 5000 koneksi, seperti pada **Gambar 5.23** berikut.



Gambar 5. 19 *Response Time FTP Zevenet Round-Robin*

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *response time* pada layanan *FTP load balancer Zevenet* dengan algoritme *round-robin* adalah 592ms untuk 1000 koneksi, 755ms untuk 2500 koneksi, dan 954ms untuk 5000 koneksi. Nilai *response time* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer Zevenet* pada layanan *FTP* dengan algoritme *round robin* menunjukkan nilai rata-rata *response time* yang cukup kecil pada koneksi rendah maupun pada koneksi tinggi.

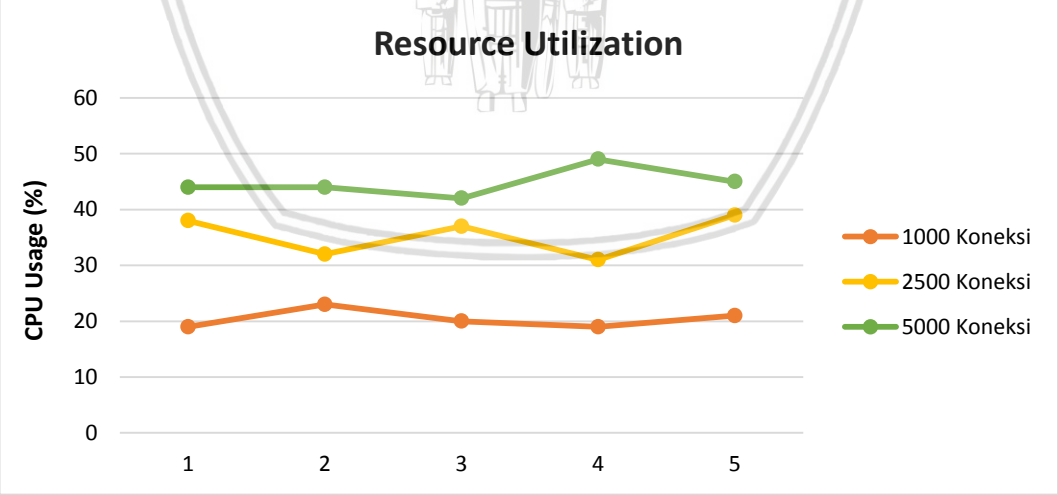
Pengujian juga dilakukan dengan mengambil nilai resource utilization. Berikut adalah tabel hasil pengujian kinerja skenario 2 layanan FTP Zevenet dengan algoritme round-robin dengan parameter resource utilization yang dilakukan sebanyak 5 kali percobaan:

Tabel 5. 14 *Resource Utilization FTP Zevenet Round Robin*

***Resource Utilization* (CPU%)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	19	38	44
2	23	32	44
3	20	37	42
4	19	31	49
5	21	39	45

Pengujian kinerja skenario 2 layanan *FTP Zevenet* dengan algoritme *round-robin* dengan parameter *resource utilization* dilakukan sebanyak 5 kali. Nilai parameter *resource utilization* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *FTP* pada *load balancer Zevenet*. *Request FTP* yang dikirimkan *client* mengunduh *file intro.pdf*. Data nilai *cpu usage* diatas diambil menggunakan *plugins Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *resource utilization* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *FTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *Zevenet* pada layanan *FTP* menggunakan algoritme *round robin* dengan parameter *resource utilization*.

Pada **Gambar 5.24** menunjukkan hasil rata-rata pengujian parameter *resource utilization* pada layanan *FTP load balancer HAproxy* dengan algoritme *round-robin*. Pengujian ini dilakukan sebanyak 5 kali percobaan. Pengambilan data dilakukan pada salah satu *server backend* dimana nilai *cpu usage* digunakan sebagai standarisasi pengujian. Pada percobaan pertama nilai *cpu usage* yang didapat adalah 19% untuk 1000 koneksi, 38% untuk 2500 koneksi dan 44% pada 5000 koneksi. Percobaan kedua didapatkan nilai *cpu usage* yang naik pada 1000 koneksi sebesar 23%, 32% pada 2500 koneksi, sedangkan pada 5000 koneksi nilai *cpu usage* stabil pada 44%. Untuk percobaan ketiga didapatkan nilai 20% pada 1000 Koneksi, 37% pada 2500 koneksi dan 42% untuk 5000 koneksi. Sedangkan percobaan keempat nilai *cpu usage* yang didapat adalah 19% untuk 1000 koneksi, 31% untuk 2500 koneksi, dan 49% untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung naik, 21% untuk 1000 koneksi, 39% untuk 2500 koneksi dan 45% untuk 5000 koneksi.



Gambar 5. 20 *Resource Utilization FTP Zevenet Round-Robin*

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *resource utilization* pada layanan *FTP load balancer Zevenet* dengan algoritme *round-robin* adalah 19% *cpu usage* untuk 1000 koneksi, 35% *cpu usage* untuk 2500 koneksi, dan 44% *cpu usage* untuk 5000 koneksi. Nilai *resource utilization* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer Zevenet* pada layanan *FTP* dengan algoritme *round robin* menunjukkan nilai rata-rata *resource utilization* yang baik karena tidak menyebabkan *overload* pada *server backend*.

5.2.2.4 Pengujian Skenario 2 *Zevenet* dengan Algoritme *Least Connection*

Pengujian skenario 2 layanan *FTP* pada *Zevenet* dengan algoritme *least connection* dilakukan untuk mendapatkan nilai performa *load balancing Zevenet* pada layanan *FTP* menggunakan algoritme *least connection* dengan parameter pengujian *response time* dan *resource utilization*.

Langkah pengujian adalah sebagai berikut:

1. *Client* mengirim *request FTP* dengan mengunduh *file intro.pdf* pada direktori */file* menggunakan *Apache Jmeter* pada *Zevenet*.
2. *Request* yang dikirimkan sebanyak 1000, 2500 dan 5000 koneksi ke alamat *FTP service load balancer* yaitu 192.168.186.10:21 dan diteruskan menuju *server backend* menggunakan algoritme *least connection*.
3. Pengujian dilakukan sebanyak 5 kali. Data dengan parameter *response time* dan *resource utilization* diambil menggunakan *plugins Apache Jmeter* dan ditampilkan dalam bentuk tabel.

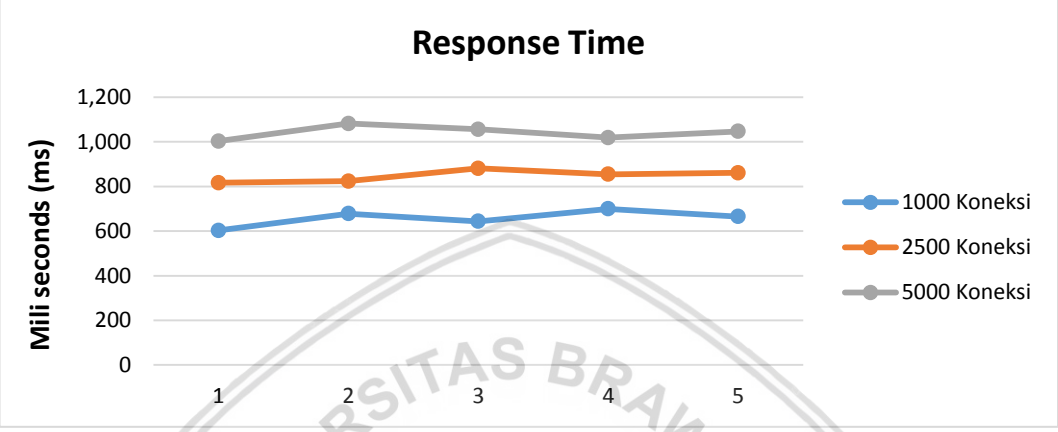
Berikut adalah tabel hasil pengujian kinerja skenario 2 layanan *FTP Zevenet* dengan algoritme *least connection* dengan parameter *response time*:

Tabel 5. 15 *Response Time FTP Zevenet Least Connection*

***Response Time* (ms)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	603	817	1.003
2	678	824	1.082
3	644	881	1.056
4	700	855	1.019
5	665	861	1.047

Pengujian kinerja skenario 2 layanan *FTP Zevenet* dengan algoritme *least connections* dengan parameter *response time* dilakukan sebanyak 5 kali. Nilai parameter *response time* diambil pada saat pengiriman *reques*t sebanyak 1000, 2500 dan 5000 koneksi *FTP* pada *load balancer Zevenet*. *Request FTP* yang dikirimkan *client* mengunduh *file intro.pdf*.

Data nilai *response time* diatas diambil menggunakan plugins *Apache Jmeter*. Data tersebut dirubah menjadi grafik dengan mengambil nilai *response time* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *FTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *Zevenet* pada layanan *FTP* menggunakan algoritme *least connections* dengan parameter *response time*.



Gambar 5. 21 *Response Time FTP Zevenet Least Connection*

Pada percobaan pertama nilai *response time* yang didapat adalah 603ms untuk 1000 koneksi, 817ms untuk 2500 koneksi dan 1003ms pada 5000 koneksi. Percobaan kedua didapatkan nilai *response time* yang naik pada 1000 koneksi sebesar 678ms, 1082ms pada 5000 koneksi, sedangkan pada 2500 koneksi nilai response time cenderung naik menjadi 824ms. Untuk percobaan ketiga didapatkan nilai 644ms pada 1000 Koneksi, 881ms pada 2500 koneksi dan 1056ms untuk 5000 koneksi. Sedangkan percobaan keempat nilai *response time* yang didapat adalah 700ms untuk 1000 koneksi, 855ms untuk 2500 koneksi, dan 1019ms untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung naik 665ms untuk 1000 koneksi, 861ms untuk 2500 koneksi dan 1047ms untuk 5000 koneksi, seperti pada **Gambar 5.25** diatas.

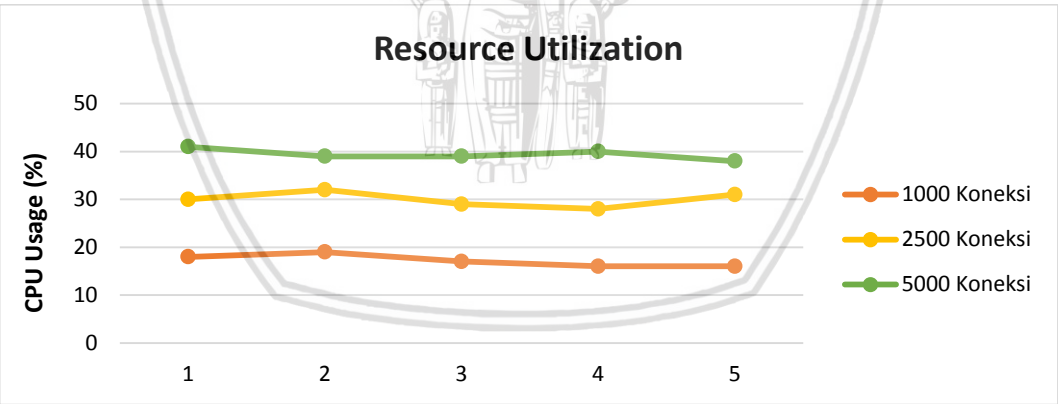
Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *response time* pada layanan *FTP load balancer Zevenet* dengan algoritme *least connection* adalah 84020ms untuk 1000 koneksi, 85680ms untuk 2500 koneksi, dan 94020ms untuk 5000 koneksi. Nilai *response time* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer Zevenet* pada layanan *FTP* dengan algoritme *least connection* menunjukkan nilai rata-rata *response time* yang lebih besar dibandingkan dengan algoritme *round-robin* pada jumlah koneksi yang besar. Hal ini disebabkan karena algoritme *least connection* memperhatikan jumlah koneksi pada setiap server.

Pengujian juga dilakukan dengan mengambil nilai *resource utilization*. Berikut adalah tabel hasil pengujian kinerja skenario 2 layanan *FTP Zevenet* dengan algoritme *least connection* dengan parameter *resource utilization* yang dilakukan sebanyak 5 kali percobaan:

Tabel 5. 16 *Resource Utilization FTP Zevenet Least Connection*

***Resource Utilization* (CPU%)**			
Percobaan	1000 Koneksi	2500 Koneksi	5000 Koneksi
1	18	30	41
2	19	32	39
3	17	29	39
4	16	28	40
5	16	31	38

Nilai parameter *resource utilization* diambil pada saat pengiriman *request* sebanyak 1000, 2500 dan 5000 koneksi *FTP* pada *load balancer Zevenet*. *Request FTP* yang dikirimkan *client* mengunduh file *intro.pdf*. Data nilai *cpu usage* diatas diambil menggunakan plugins *Apache Jmeter*. Data tersebut akan dirubah menjadi grafik dengan mengambil nilai *resource utilization* rata-rata setiap percobaan pada masing-masing 1000, 2500 dan 5000 koneksi *FTP* yang dikirimkan *client*. Kemudian dilakukan analisa terhadap grafik tersebut untuk menarik kesimpulan terhadap kinerja *Zevenet* pada layanan *FTP* menggunakan algoritme *least connection* dengan parameter *resource utilization*.



Gambar 5. 22 *Resource Utilization FTP Zevenet Least Connection*

Pada **Gambar 5.12** menunjukkan hasil rata-rata pengujian parameter *resource utilization* pada layanan *FTP load balancer Zevenet* dengan algoritme *least connection*. Pengujian ini dilakukan sebanyak 5 kali percobaan. Pengambilan data dilakukan pada salah satu *server backend* dimana nilai *cpu usage* digunakan sebagai standarisasi pengujian. Pada percobaan pertama nilai *cpu usage* yang didapat adalah 18% untuk 1000 koneksi, 30% untuk 2500 koneksi dan 41% pada 5000 koneksi.

Percobaan kedua didapatkan nilai *cpu usage* yang naik pada 1000 koneksi sebesar 19%, 39% pada 5000 koneksi, sedangkan pada 2500 koneksi nilai *cpu usage* naik menjadi 32%. Untuk percobaan ketiga didapatkan nilai 17% pada 1000 Koneksi, 29% pada 2500 koneksi dan 39% untuk 5000 koneksi. Sedangkan percobaan keempat nilai *cpu usage* yang didapat adalah 16% untuk 1000 koneksi, 28% untuk 2500 koneksi, dan 40% untuk 5000. Pada percobaan terakhir didapatkan nilai yang cenderung naik, 16% untuk 1000 koneksi, 31% untuk 2500 koneksi dan 38% untuk 5000 koneksi.

Dengan menganalisa grafik dan penjelasan diatas dapat disimpulkan bahwa nilai rata-rata *resource utilization* pada layanan *FTP load balancer Zevenet* dengan algoritme *least connection* adalah 16% *cpu usage* untuk 1000 koneksi, 30% *cpu usage* untuk 2500 koneksi, dan 39% *cpu usage* untuk 5000 koneksi. Nilai resource utilizati*on* berubah seiring dengan perubahan jumlah koneksi yang dikirimkan. Proses pendistribusian *request* yang dilakukan *load balancer Zevenet* pada layanan *FTP* dengan algoritme *least connection* menunjukkan nilai rata-rata *resource utilization* yang lebih kecil dibandingkan dengan algoritme *round-robin* pada jumlah koneksi banyak dan cenderung sama pada jumlah koneksi sedikit.

Pada skenario pengujian layanan *FTP*, *HAproxy* menampilkan nilai rata-rata *response time* yang lebih baik pada setiap algoritme, yakni 309ms, 463ms, 654ms untuk algoritme *round robin* dan 328ms, 474ms, 752ms untuk algoritme *least connection*. Sedangkan pada skenario pengujian layanan *FTP, Zevenet* menampilkan nilai yang lebih baik pada setiap algoritme dengan nilai rata-rata *CPU usage* nya 19%, 35%, dan 44% untuk algoritme *round-robin* dan 16%, 30%, dan 39% untuk algoritme *least connection*.

BAB 6 PENUTUP

6.1 Kesimpulan

Dari hasil implementasi, pengujian dan analisis sistem *load balancing* yang dibangun, penulis dapat menyimpulkan bahwa:

1. Pengimplementasian *multi service load balancing* dibangun dengan cara membuat *HAproxy* dan *Zevenet* mampu menjadi penghubung antara *client* dan *server-server multi service* serta mendistribusikan *traffic* data berdasarkan *algoritma round robin* dan *least connection* pada *service HTTP* dan *FTP*. Dalam implementasinya dilakukan pengujian kinerja menggunakan 2 skenario.
2. Perbandingan kinerja *multi service load balancing* menggunakan *HAproxy* dan *Zevenet* dilakukan dengan menjalankan skenario pengujian pada layanan *HTTP* dan *FTP* dengan parameter pengujian *response time* dan *resource utilization*. Pada layanan *HTTP* dan *FTP*, *HAproxy* menunjukkan nilai hasil pengujian kinerja yang lebih baik pada parameter *response time*. Sedangkan *Zevenet* menunjukkan nilai hasil pengujian kinerja yang lebih unggul pada parameter *resource utilization* untuk layanan *HTTP* dan *FTP*.

6.2 Saran

Saran dari Penulis untuk pengembangan *multi-service load balancing* selanjutnya adalah:

1. Parameter pengujian yang digunakan tidak hanya *response time* dan *resource utilization* serta membandingkan hasil pengujian algoritme *round-robin* dan *least connection* dengan algoritme *load balancing lainnya*.
2. Pengembangan *multi-service load balancing* pada *service* lainnya dan berfokus pada sisi keamanan sistem.

DAFTAR PUSTAKA

Bourke, Tony. 2001. Server Load Balancing. O'Reilly & Associates Inc. [online] Tersedia di < index-of.es/Networking/Server%20Load%20Balancing.pdf > [Diakses 10 Oktober 2017]

Chauhan, Yogesh. Chauhan, Siva. 2012. Performance Comparison of Server Load Distribution with FTP and HTTP. [online] Tersedia di < HTTPs://pdfs.semanticscholar.org/5c61/acac0795e563a49a93279f446e4b9b1f046c.pdf > [Diakses 8 Oktober 2017]

Haproxy. Load Balancing. [online] Tersedia di < HTTPs://www.haproxy.com/solutions/load-balancing/ > [Diakses 8 Oktober 2017]

Hunt, Craig . 2002. TCP/IP Network Administration. O'Reilly Media, Inc . [online] Tersedia di < HTTPs://www.cipl.net.in/courses/PDF/TCPIP.pdf > [Diakses 8 Oktober 2017]

IBM. 2011. Load Balancer for IPv4 Administration Guide. IBM Corporation. [online] Tersedia di <FTP://170.225.15.40/software/library/LBguide_ipv4.pdf> [Diakses 10 Oktober 2017]

Kopparapu, Chandra. 2002. Load Balancing Servers, Firewalls, and Caches. John Wiley & Sons, Inc. [online] Tersedia di < HTTP://box.cs.istu.ru/public/docs/other/_Unsorted/new/Misc/Load%20Balancing%20Servers,%20Firewalls%20and%20Caches%20(2002,%20Wiley).pdf > [Diakses 10 Oktober 2017]

RedHat. 2017. Red Hat Enterprise Linux 6 Load Balancer Administration. Red Hat, Inc. [online] Tersedia di < HTTPs://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/pdf/load_balancer_administration/Red_Hat_Enterprise_Linux-6-Load_Balancer_Administration-en-US.pdf > [Diakses 10 Oktober 2017]

Sharma, Sandeep. Singh, Sarabjit. 2008. Performance Analysis of Load Balancing Algorithms. [online] Tersedia di < HTTPs://waset.org/publications/5537/performance-analysis-of-load-balancing-algorithms > [Diakses 10 Oktober 2017]

Zevenet. Community Edition v3.05 Administration Guide. [online] Tersedia di < HTTPs://www.zevenet.com/knowledge-base_category/community-edition-v3-05-administration-guide/ > [Diakses 10 Oktober 2017]