ANALISIS *OPEN FLOW LOAD BALANCING WEB SERVER* DENGAN ALGORITMA *LEAST CONNECTION* PADA *SOFTWARE DEFINED NETWORK*

Erine Karantha Denny L.[1], Rakhmadhany Primananda, S.T., M.Kom. [2], R. Arief Setyawan, S.T., M.T. [3]

Program Studi Teknik Komputer, Program Teknologi Informasi Dan Ilmu Komputer, Universitas Brawijaya

Jl. Veteran No. 8 Malang, Gedung A PTIIK-UB

Email : karanthadenny@gmail.com[1], rakhmadhany@yahoo.com[2], rarief@ub.ac.id[3]

ABSTRAK

Pada perkembangan jaringan komputer saat ini terdapat sebuah teknologi dimana sistem pengontrol dari arus data dipisahkan dari perangkat kerasnya. *Software defined network* merupakan jaringan komputer yang memisahkan antara *control plane* dan data *plane* pada *switch. Software defiened network* juga merupakan jaringan komputer yang sangat *flesibel* karena dikonfigurasikan dan dikendalikan melalui sebuah *software* terpusat. Cara komunikasi antara perangkat dan kontroller menggunakan sebuah protokol yang disebut dengan *Openflow*.

Sebagai konsep jaringan yang sangat komplek *software defined network* menawarkan *scability* dan *programmability* seperti *load balancing web server. Load balancing* merupakan sebuah metode pembagian beban yang diberikan ke suatu *web server* ketika melayani *request* yang dilakukan oleh user. Pembagian beban ke *web server* dilakukan berdasarkan dengan algoritma *load balancing. Least connection* merupakan algoritma *load balancing* yang akan menyalurkan koneksi jaringan ke *server* yang memiliki koneksi aktif paling sedikit.

Melalui Implementasi sistem *load balancing* dengan algoritma *least connection* sebagai pembagi beban ke *server*. Sistem *load balancing* memberikan nilai rata – rata koneksi per detik yang cukup rendah yakni 1.28 conns/s dan akan turun ketika koneksi yang di lakukan *user* semakin tinggi. Berbeda dengan nilai *Reply time* yang akan meningkat ketika jumlah permintaan *user* juga semakin tinggi dengan nilai rata- rata 199.58 ms. Untuk nilai *error* memberikan hasil yang baik dengan nilai error 0 yang berarti tidak terjadi error dari beberapa skenario pengujian pada sistem.

Kata kunci: *load balancing, web server, least connection, software defined network*.

1. LATAR BELAKANG

Teknologi saat ini berkembang sangat pesat khususnya pada pemanfaatan jaringan Internet. Salah satu satunya software define network (SDN). *Software defined networking* adalah sebuah pendekatan jaringan komputer dengan sistem pengontrol dari arus data dipisahkan dari perangkat kerasnya. Perangkat networking secara umum terdiri dari dua bagian yaitu control plane dan data plane. Control plane merupakan bagian yang berfungsi sebagai pengatur logika pada perangkat, sedangkan data plane berfungsi untuk meneruskan paket yang masuk ke suatu port menuju port lainnya dengan komunikasi pada control plane.

Pada *software defined network* control jaringan dipisahkan dari sistem forwadingnya dan controller tersebut dapat kita program secara langsung. Komunikasi antara perangkat dan controller menggunakan sebuah protocol yang disebut dengan Openflow.

Konsep jaringan *software defined network* menawarkan *scability* dan program *ability* untuk penggunaan jaringan yang semakin komplek seperti *load balancing web server* (Senthil, Ranjani S., 2015). *Load balancing* merupakan metode pembagian beban yang diberikan ke suatu web server. Ketika suatu situs web memiliki kunjungan yang tinggi maka beban kinerja server juga meningkat, sehingga beban server harus dibagi dengan server yang lain agar kinerja server menjadi lebih optimal. Dalam melakukan pendistribusian ke web server *load balancing* memiliki beberapa algoritma, salah satunya adalah *least connection*. *Load balancing least connection* ini akan menyalurkan koneksi jaringan ke server yang memiliki koneksi aktif paling sedikit. Pada server yang memiliki kemampuan pemrosesan yang sama, *least connection* akan mendistribusikan beban permintaan dengan baik karena permintaan yang panjang tidak akan di salurkan ke sebuah server (Yogi, Kurniawan.,2014).

Kinerja dari web server dipengaruhi oleh permintaan yang di lakukan oleh user. Semakin banyak permintaan terhadap suatu halaman web maka beban terhadap web server akan semakin tinggi sehingga respon yang akan di berikan juga semakin lambat dan dapat menyebabkan server utama mati. Sehingga diperlukan penambahan server untuk tujuan melakukan pembagian beban server mengunkan algoritma least connection agar tidak terjadi overload.

Algoritma least connection juga dapat mendistribusikan beban permintaan dengan baik karena permintaan yang panjang tidak akan di salurkan ke sebuah server. Dengan demikian

sistem load balancing dengan mengunakan algoritma least connection ini dapat memberikan solusi terhadap beban server dalam memberikan respon permintaan user.

Berdasarkan penjelasan di atas maka diperlukan pengujian terhadap sistem load balancing pada software defined network dengan menggunakan metode algoritma least connection.

2. DASAR TEORI

2.1 Software Defined Network

Software defined network adalah suatu arsitektur jaringan dimana control *network* dipisahkan dari *system forwadingnya* dan *controller* dapat diprogram secara langsung. Pada umumnya perangkat jaringan yang ada saat ini terdiri dari *control plane* dan data *forwading plane*. Keduanya tertanam dalam satu perangkat. Sedangkan dalam SDN antara *control plane* dan *forwading planenya* dipisahkan dalam suatu perangkat yang berbeda. Konsep dasar SDN adalah dengan melakukan pemisahan eksplisit antara control dan *forwarding plane*, serta kemudian melakukan abstraksi sistem dan mengisolasi kompleksitas yg ada pada komponen atau subsistem dengan mendefinisikan antar muka (*interface*) yang standard. Control layer memegang peran penting mengendalikan system jaringan, setiap perangkat dari berbagai macam vendor dapat beroprasi sesuai perintah dari controllernya. Cara komunikasi antara perangkat dan controllernya sendiri menggunakan suatu protocol yang disebut dengan OpenFlow. OpenFlow sendiri bisa diibaratkan seperti sebuah CPU pada computer. (Astuto, et al. 2014).

2.2 Openflow

Openflow memungkinkan akses langsung dan dapat digunakan untuk memanipulasi forwarding pada perangkat jaringan seperti *switch* dan *router*, baik secara fisik maupun virtual seperti menambah, menghapus maupun melakukan modifikasi pada *flow table*. Mekanisme kerja dari protokol *openflow* adalah pada saat *switch openflow* menerima sebuah paket dan tidak memiliki kecocokan dengan tabel *flow* yang sudah ada maka *switch openflow* akan meneruskan paket menuju ke *controller openflow*. Kemudian Kontroller akan memberikan respon kepada paket yang datang tersebut berdasarkan tabel flow. Sehingga komunikasi yang terjadi antara control plane dengan data plane melalui *controller* bisa memberikan respon yang sesuai terhadap setiap paket yang datang.

2.3 Controller

Controller berfungsi sebagai pusat control dan logika jaringan yang dapat membantu melakukan komunikasi dan konfigurasi antara application layer dan infrastrukture layer. Implementasi load balancing controller yang digunakan adalah POX controller yang berfungsi untuk mengatur aplikasi yang ada pada jaringan. Setting komunikasi openflow interface POX controller ini menggunakan bahasa pemograman python. Pada POX controller juga tersedia support modules untuk openflow. Untuk menangani konektifitas dan juga metode untuk berinteraksi antara application interface dengan openflow pada POX juga terah tersedia.(lara et al, 2013).

2.4 Mininet

Mininet adalah merupakan sebuah Emulator jaringan yang mensimulasikan koleksi dari host-end, switch, router, dan link pada single kernel Linux. Masing-masing elemen ini disebut sebagai "host" menggunakan virtualisasi ringan untuk membuat sistem tampilan tunggal sehingga terlihat seperti jaringan yang lengkap, menjalankan kernel yang sama, sistem, dan user code. Mininet penting bagi komunitas open-source SDN. Mininet biasanya digunakan sebagai simulasi, verifikasi, testing tool, dan resource. (O'Reilly, 2013).

2.5 Load Balance

Load balancing adalah teknik untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal, memaksimalkakn throughput, memperkecil waktu tanggap dan menghindari overload pada salah satu jalur koneksi (Sirajuddin,2012).

2.6 Least Connection

Mekasnisme server menggunakan algoritma least connection akan menyalurkan koneksi jaringan kepada server yang memiliki koneksi aktif paling sedikit. Pada server yang memiliki kemampuan pemrosesan yang sama, algoritma penjadwalan least connection akan mendistribusikan beban permintaan dengan baik karena permintaan yang panjang tidak akan disalukan kepada server (Yogi Setiawan,2014).

2.7 Web Server

Web server dapat di artikan sebagai sebuat perangkat keras ataupun perangkat lunak yang menyediakan layanan kepada pengguna web dengan menggunakan protokol HTTP ataupun HTTPS atas berkas-berkas yang terdapat dalam sebuah situs web dalam layanan ke pengguna menggunakan aplikasi tertentu seperti web browser. Penggunaan paling umum dari sebuah web server adalah untuk menempatkan sebuah situs web, namun pada kenyataannya penggunaannya dapat diperluas sebagai tempat

penyimpanan data ataupun untuk menjalankan sejumlah aplikasi bisnis (Kadir,2003).

2.8 Httperf

Dalam melakukan pengujian jaringan banyak tool yang dapat digunakan salah satunya adalah httperf. Httperf sendiri merupakan sebuah tool yang dapat digunakan untuk melakukan pengujian terhadap suatu web server dengan menggunakan beberapa parameter tertentu antara lain reply rate, connection rate, error rate dan juga standart deviasi pada suatu web server (David M, 1998).

3. METODOLOGI

3.1 Studi Literatur

Pada bagian ini membahas tentang dasar teori yang mendukung segala kebutuhan dalam mengimplementasikan load balancing pada software defined network dengan metode algoritma *least connection*. Adapun yang dapat dijadikan bahan dalam studi literatur meliputi *Software defined network, controller,* dan juga *web server*.

3.2 Analisa Kebutuhan

Analisis kebutuhan yang diperlukan untuk menganalisis apa saja yang di butuhkan oleh sistem pada penelitian yang dilakukan, sehingga dapat dilakukan sesuai dengan yang diharapkan. Pada sistem ini terdiri dari kebutuhan sistem dan juga kebutuhan software. Kebutuhan sistem antara lain :

1. Sistem dapat melakukan service terhadap client dengan jumlah request tertentu dalam satu waktu.
2. Sistem dapat melakukan pembagian beban server berdasarkan algoritma least connection.
3. Sistem dapat melakukan pengontrolan terhadap web server secara terpusat melalui controller pada software defined network.
4. Sistem dapat melakukan log terhadap jumlah koneksi setiap server dan juga log pada koneksi yang flow.

Kebutuhan *software* antara lain :

1. Kebutuhan perangkat lunak antara lain mininet-VM dengan operating system Ubuntu 14.04, Xming, Putty, python, pox controller dan httperf.

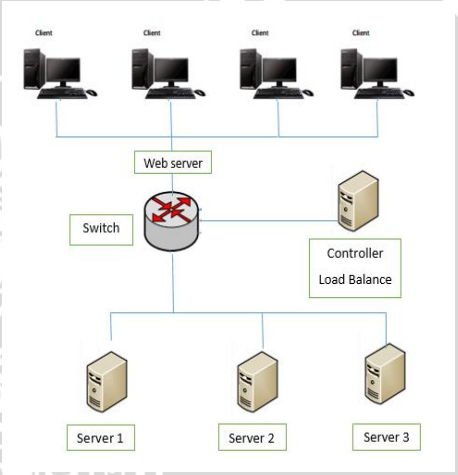
3.3 Pengujian dan Analisis

Pengujian dan analisis sistem dilakukan untuk mengetahui kinerja. Pada penelitian ini terdapat dua pengujian untuk menguji keberhasilan sistem:

1. Melakukan request dengan menggunakan 4 client secara bersamaan dengan memberikan beban pada server 50, 75, 100 dan 150 request/sec.
2. Melakukan request dengan menggunakan 4 client secara bersamaan dengan memberikan rate 25, 50, 75, dan 100 connect/sec.
3. Melakukan request dengan memberikan beban server dengan 200 request/sec dan 400 request /sec
4. Melakukan analisis dengan parameter yang digunakan adalah connection rate, reply time, cpu time dan error.
5. Melakukan analisis beban server saat awal koneksi, ketika terjadi expired flow, dan setelah expired flow.

3.4 Perancangan Sistem

Pada bagian ini dijelaskan perancangan sistem dari tahap pembuatan topologi sampai hasil pengujian dan kesimpulan yang dilakukan pada sistem.

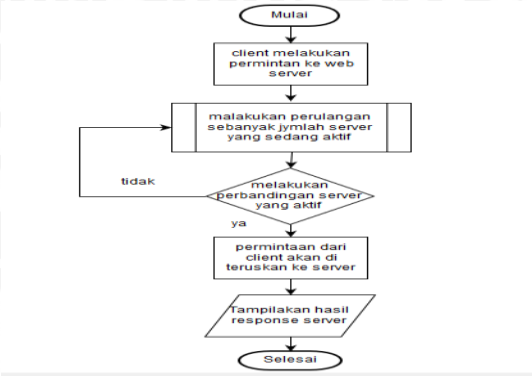


Gambar 3.2 Perancangan Sistem

Pada diatas merupakan perancangan yang digunakan pada topologi mininet server dengan menggunakan tiga server virtual yang masing-masing akan berkomunikasi pada port 80. Kemudian Switch yang bertugas meneruskan paket ke controller untuk selanjutnya diberikan action terhadap paket tersebut, lalu *traffic* paket akan dibagi berdasarkan algoritma least connection. Setelah itu dilakukan pengujian performa dan kinerja dari web server dengan *client* melakukan request. Perancangan sistem sendiri terdiri dari tujuh host dengan tiga host sebagai server dan empat host sebagai client.

3.5 Algoritma Load Balancing

Berikut algoritma yang menjelaskan tentang load balancing yang akan diterapkan:

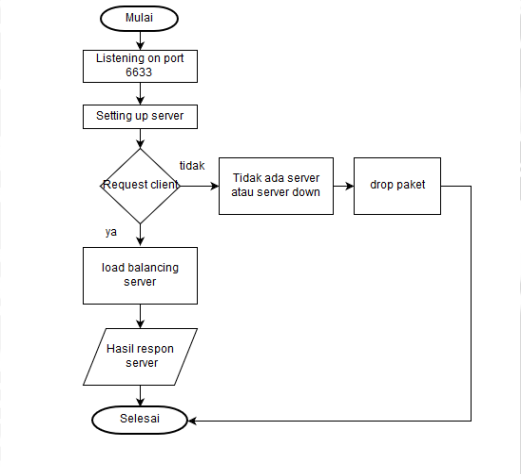


Gambar 3.3 Algoritma Load Balancing

Pada diagram diatas proses pembagian traffic dawali dengan request client ke server yang selanjutnya request tersebut akan dilakukan pengecekan untuk pemilihan traffic jaringan. Selanjutnya setelah melakukan pengecekan, server yang memiliki sedikit permintaan maka paket akan di teruskan ke traffic jaringan yang masuk ke server tersebut. Namun ketika keadaan server tidak dalam melanyani request dari client maka request dari client yang pertama kali masuk akan di teruskan ke server yang pertama.

3.6 Alur Sistem

Load balancing pada software defined network dengan menggunakan algoritma least connection akan di rancang alur sistem sebagai berikut :

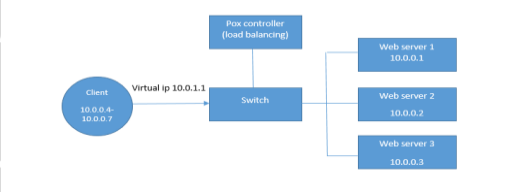


Gambar 3.4 Alur Sistem

Ketika sistem dijalankan maka akan melakukan listening ke port 6633 dan selanjutanya melakukan setting up server. Ketika ada request dari client sistem akan melakukan pengecekan terhadap server yang sedang up

Pada saat tidak ada server yang up atau aktif maka request dari client akan di drop dari traffic, tetapi jika semua pengecekan menunjukan ada server yang sedang up maka request dari client akan di lanjutkan ke server. Sebelum sampai ke server traffic yang masuk akan di bagi berdasarkan algoritma load balancing least connection. Setelah melakukan pengecekan server dengan load balancing maka selanjutnya trafiic dari client akan di teruskan ke server yang selanjutnya akan menunggu respon dari server yang dituju untuk kemudian ditampilkan respon server tersebut.

3.7 Alur Komunikasi



Gambar 3.5 Alur Komunikasi

Alur komunikasi yang akan lakukan oleh user. Pertama user dengan alamat client 10.0.0.1 sampai 10.0.0.7 akan mengakses ip virtual web server dengan alamat 10.0.1.1. Selanjutnya client akan terhubung ke switch yang bertugas untuk meneruskan permintaan user menuju ke alamat destination. Switch akan terhubung ke controller, controller bertugas untuk membuat keputusan terhadap paket yang masuk. Kemudian sistem load balancing akan membagi request user berdasarkan jumlah server yang tersedia.

4. IMPLEMENTASI

4.1 Membuat Topologi

Untuk melakukan implementasi load balancer di web server pada software defined network, hal pertama yang dilakukan adalah melakukan Inisialisasi topologi pada mininet. Topologi yang akan digunakan adalah dengan menggunakan tujuh *host* dan pox sebagai remote controller serta *single* switch.

```
1  $ sudo mn --topo single,7 --mac --arp --controller=remote
```

Membuat topologi dengan single switch dengan jumlah host sebanyak tujuh host, Kemudian mengatur alamat MAC address secara otomatis dan juga protokol ARP antar host dan remote contoller yang akan digunakan sebagai pengatur ttraffic paket. Kemudian menambahkan controller yaitu (c0) dan single switch (s1). Untuk mengubungkan masing-masing host juga dilakukan create links dari (h1,s1) sampai (h7,s1).

4.2 Running Load Balancing

Untuk melakukan implementasi load balancing pada *software defined network* dengan menggunakan algoritma least connection terlebih dahulu dilakukan running sistem load balancing dengan menggunakan *pox* sebagai *controller*.

```
$ ./pox.py log level DEBUG misc.ip_loadbalancing --ip=10.0.1.1
--servers=10.0.0.1,10.0.0.2,10.0.0.3
```

Ketika controller pox up maka controller akan melakukan listening pada port 6633 dan akan melakukan setting up pada seerver 10.0.0.1 sampai 10.0.0.3. Maka sistem load balncing siap untuk dijalankan.

4.3 Setting Server

Untuk melakukan pengaturan *server* pada *host* digunakan perintah sebagai berikut :

```
#python -m SimpleHTTPServer 80
```

Pengaturan server pada host sudah selesai dan server up siap digunakan pada protokol HTTP port 80.

4.4 Setting Client

Untuk malakukan pengujian load balancing pada web server di perlukan client untuk melakukan request. Berikut program untuk melakukan setting client.

```
xterm h1 h2 h3 h4 h5 h6 h7
```

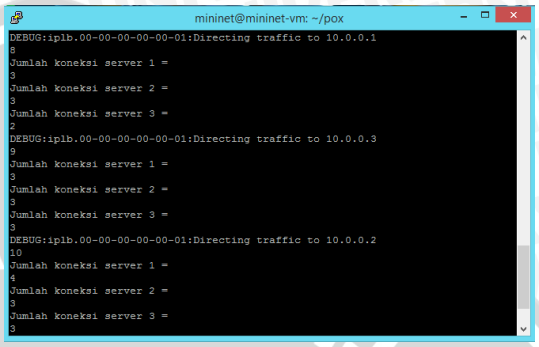
Pada program diatas xterm h1 sampai dengan h7 digunakan untuk membuat jaringan host h1 sampai h7 yang nantinya akan digunakan sebagai server dan client. Program diatas adalah tampilah jaringan host h4 sampai h7. Host h4 sampai h7 ini nantinya akan berfungsi sebagai client 1 sampai 4 yang akan melakukan permintaan ke web server.

4.5 Web Server

Gambar 4.1 Web Server

Untuk malakukan implementasi load balancing digunakan perintah #curl pada alamat IP 10.0.1.1 yang dilakukan pada sisi client. Kemudian akan muncul halaman doctype html yang nantinya data tersebut digunakan untuk melakukan permintaan ke server.

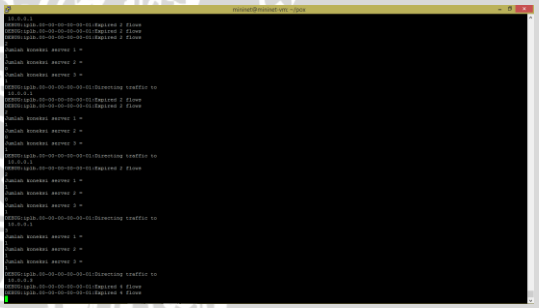
4.6 Sistem Load balancing Least Connection



Gambar 4.2 Load Balancing

Perbandingan jumlah koneksi dari masing-masing server. Server yang memiliki jumlah koneksi paling sedikit maka akan di berikan beban koneksi berikutnya.

4.7 Expired Flow

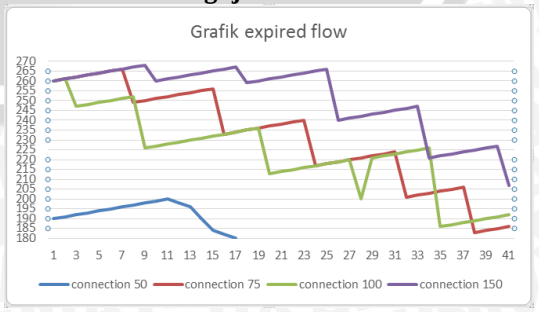


Gambar 4.3 Expire flow

Server yang memiliki jumlah koneksi terbanyak yang akan di cek terlebih dahulu untuk di berikan expire flow.

5. PENGUJIAN

5.1 Skenario Pengujian 1



Gambar 5.1 Grafik flow

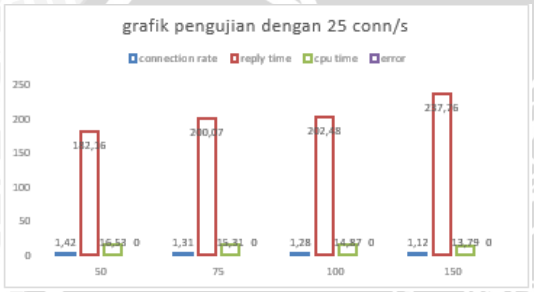
Berdasarkan gambar diatas dapat di simpulkan ketika user melakukan *request* ke

server terjadi dua kondisi yakni kondisi ketika request sedang up atau diproses server dan kondisi ketika request down atau di drop dari traffic oleh sistem sehingga terbentung grafik.

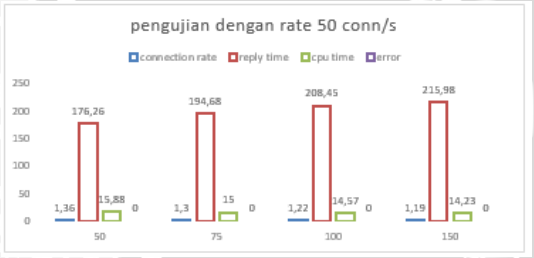
Dilihat dari hasil pengujian ketika terjadi 200 request tidak terjadi expired sehingga saat request sudah selesai diproses maka koneksi akan menurun sampai nilai beban server menjadi 0 kembali. Kemudian pada request 75 terjadi expired pada koneksi ke -266, pada request 100 terjadi expired pada koneksi ke -261 dan pada request 150 terjadi expired pada koneksi ke -268 sehingga terbentuk lah grafik seperti diatas.

5.2 Pengujian dengan menggunakan rate

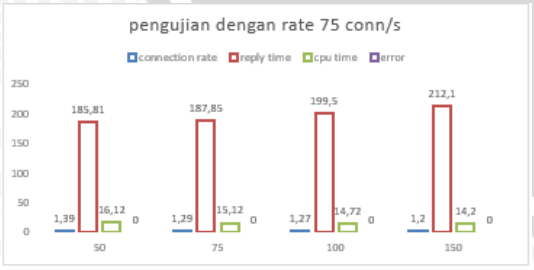
Pada pengujian ini dilakukan dengan memberikan rete 25 conn/s dengan *request user* sebanyak 50 kali, 75 kali, 100 kali dan 150 kali



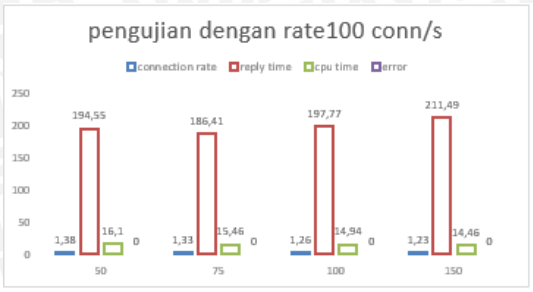
Gambar 1



Gambar 2



Gambar 3

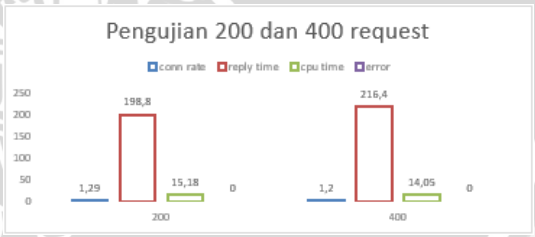


Gambar 4

Berdasarkan gambar 1, 2, 3 dan 4 pengujian diatas ketika jumlah request yang dilakukan semakin banyak maka nilai dari *connection rate* juga akan turun dari 4 kali pengujian nilai *connection rate* rata-rata turun sebesar 0.1 conns/s. begitu juga dengan *cpu time*, semakin banyak jumlah request yang di laukukan nilai dari *cpu time* juga akan semakin kecil. Dari 4 kali pengujian rata- rata nilai *cpu time* turun sebesar 0.9 s pada rate 50 kemudian 0.5 s pada rate 75, pada rate 100 sebesar 0.64 s dan 0,54 s pada rate 150. Berbeda dengan *reply time* nilainya terus naik seiring dengan semakin banyak jumlah request yang dilakukan. Pada percobaan yang dilakukan nilai *reply time* rata-rata naik sebesar 13.75 ms dan nilai error yang terjadi adalah 0 dari pengujian yang dilakukan.

5.3 Pengujian dengan 200 dan 400 request

Pada skenario pengujian ini dilakukan jumlah permintaan *client* terhadap *server* yang melebihi kapasitas beban *server* :



Gambar 5.1 Screenshot 200 dan 400 request

Berdasarkan diatas dapat diketahui dari hasil pengujian dengan 200 request dan 400 request tidak terjadi error pada sistem. Nilai error pada sistem load balancing tetap bernlai baik yakni menunjukan nilai 0. Hal ini membuktikan bahwa sistem masih bisa memproses request yang dilakukan *client.* Hal ini di karenakan setiap server tidak diberikan nilai limit atau batas maksimal koneksi yang mampu di proses setiap server.

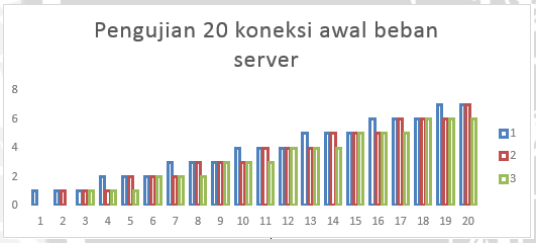
Pada *connection rate* untuk pengujian pada request 200 dan 400 nilai yang diberikan yakni 1.29 conns/s dan 1.2 conns/s lebih rendah dibandingkan nilai request 50, 75 ,100 dan 150.

Hal ini menunjukan nilai request yang semakin besar maka nilai *connection rate* akan semakin kecil.

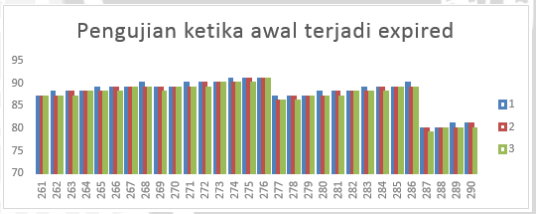
Untuk *reply time* pada pengujian dengan jumlah request 200 nilai yang di peroleh adalah 198.8 ms sedangkan dengan jumlah request 400 bernilai sebesar 216.4 ms. Sehingga semakin besar jumlah request yang di berikan maka nilai dari *reply time* juga akan semakin besar. *Cpu time* pada grafik menunjukan nilai yang menurun yakni 15.18 s pada request 200 dan 14.05 pada request 400. Nilai *cpu time* akan semakin kecil jika jumlah request yang di berikan semakin besar.

5.4 Skenario Pengujian 2

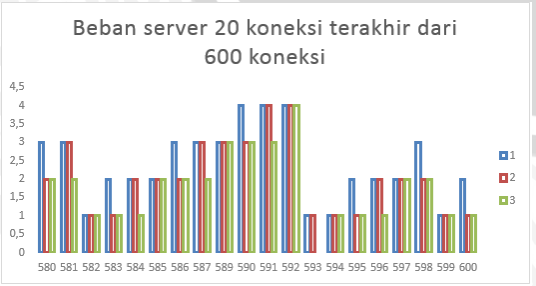
Pada skenario pengujian kedua dilakukan untuk mengetahui beban *server* terhadap *request* yang di lakukan *client* pada *web server*. Melalui skenario ini dapat didapatkan hasil pembagian beban *server* pada setiap *request* yang dilakukan oleh *client.* Pada pengujian ini jumlah request yang di berikan ke *web server* sebanyak 50 kali, 75 kali, 100 kali dan 150 kali



Gambar 1



Gambar 2



Gambar 3

Dari Gambar 1 hasil pengujian diatas dapat di jelaskan pada koneksi 1 sampai dengan 20 pembagian beban *server* dilakukan dengan cara bergantian pada *server* yang sedang aktif mulai dari server 1 sampai dengan *server* 3. Pembagian server akan terus bergantian sampai terjadi *expired flow* pada salah satu *server* yang sedang aktif

Pada gambar 2 hasil dipengujian diatas dapat dilihat ketika koneksi ke -261 sampai dengan koneksi ke -268 pembagian beban *server* dilakukan bergantian dari *server 1* sampai dengan *server 3*, beban *server* 1 adalah 90 koneksi beban *server* 2 adalah 89 koneksi dan beban *server* 3 adalah 89 koneksi. Ketika pada koneksi ke -269 terjadi expired flow beban server menjadi 89 koneksi untuk *server 1, 89 koneksi* untuk *server* 2 dan 88 koneksi untuk *server* 3. Pada hal ini algoritma *least connection* berperan dalam pemilihan *server*, *server* yang memiliki jumlah koneksi paling sedikit akan diberikan beban oneksi selanjutnya yang akan masuk. Sehingga pada koneksi ke -270 beban koneksi di berikan pada *server* ke 3 karena memiliki beban paling sedikit

Pada gambar 3 Pada koneksi ke -580 sampai konkesi ke -600 sering terjadi expired flow karena pada koneksi di atas koneksi ke -270 akan terjadi pengecekan expired flow setiap 5 detik sekali. Semakin banyak jumlah *request* yang diberikan oleh *client* maka semakin banyak expired yang terjadi dan beban *server* akan semakin sedikit.

6. KESIMPULAN DAN SARAN

6.1 Kesimpulan

1. Penerapan Algoritma least connection terhadap pengujian load balancing pada *software defined network* dapat berjalan dengan baik dan membantu meringankan beban server dalam memberikan perlayan terhadap permintaan yang dilakukan oleh user
2. Pada sistem load balancing dengan algoritma *least connection* mampu memberikan performa yang baik terhadap server. Berdasarkan pengujian yang telah dilakukan algoritma *least connection* mampu membagi *traffic* jaringan dengan memberikan beban pada server yang memiliki jumlah permintaan *user* paling sedikit.
3. Pada skenario pengujian didapatkan nilai expire flow yang berbeda di setiap pengujian. Pada pengujian 50 kali tidak terjadi *expire flow*, pengujian 75 kali *expire flow* terjadi pada koneksi ke -266,

pengujian 100 kali *expire flow* pada koneksi ke -261, dan pada pengujian 150 kali *expire flow* pada koneksi ke -268.

4. Semakin banyak jumlah request yang dilakukan maka nilai dari *connection rate* juga semakin kecil. Nilai rata- rata *connection rate* dari 4 kali pengujian 1.28 conns/s.
5. Nilai *reply time* akan semakin tinggi seiring jumlah request yang dilakukan oleh user juga semakin tinggi. Rata- rata nilai *reply time* dari semua pengujian yang dilakukan sebesar 199.58 ms.
6. Nilai *cpu time* juga akan berkurang ketika jumlah request yang dilakukan oleh user juga semakin tinggi. Dari haril pengujian didapat nilai rata- rata *cpu time* 15.08 s.
7. Pada pengujian yang dilaukan tidak terjadi error baik pada rate 25, 50, 75 maupun 100. Hal ini di karenakan sistem tidak mengalami overload karena permintaan yang sangat banyak dari user.
8. Pada pengujian dengan jumlah request 200 dan 400 sistem tidak mengalami error. Hal ini berarti sistem load balancing bekerja dengan baik walaupun request yang diberikan melebihi kapasits sistem.

2. DAFTAR PUSTAKA

Cui Chen-Xiao and Xu Ya-bin. 2016. *Research on Load Balance Method in SDN*. Internasional Journal of Grid Distributed Computing, Volume 9. No. 1 pp.25-36

Dependra, Dhakal., Bishal, Pradhan., Sunil, Dhimal., 2016. Campus Network Using Software defined network. *International Journal Of Computer And Information Technology*, Volume 138 - no 4

Kurniawan, Yogi., 2013. *Analisis Kinerja Algoritma Load Balancer dan Implementasi pada Layana Web*. Malang: Universitas Brawijaya

Lara, A., Kolasani, A., Ramamurthy, B., 2013. Network Innovation Using Open*Flow*: A Survey.*IEEE Commun. Surv. Tutor. 16*, p.1–20

Mukundha., Dr. Chinthagunta., P. Gayatri., Dr.I.Surya, Prabha., 2016. *Load Balance Scheduling Algorithm for Serving of Requests inCloud Network Using Software defined networks.* International Journal of Applied Engineering Research. (ISSN 0973-4562)

Mustafa E. M., Amin Mubark I., 2015. Load Balancing Algorithms Round-Robin, Least-Connection And Least Loaded Efficiency. *International Journal Of Computer And Information Technology*, Volume 04-Issue 02

POX, 2016. Pox controller Tutorial, Tersedia di : < http://sdnhub.org/tutorials/pox/> [Diakses 21 juni 2016]

Qingwei Du and Huaidong Zhuang. 2015. OpenFlow-Based Dynamic Server Cluster Load Balancing with Measurement Support. *Journal Of Communication*, No. 8

Senthil, Ranjani S., 2015. *Load balancing Using Software Defined Networks*. International conference on current trends in advances computing. ICCTAC.SRM University, Chennai

Sukhveer Kaur., Japinder S., Navtej Singth G., 2014. Network Programmability Using Pox Controller. *International Conference on communication, computing and sistem.*

Yuanhao, Zhou.; Li, Ruan.; Rui, Liu. 2014. A Method for *Loadbalancing* based on Software Defined Network, p. *45*