Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 4, No. 3, Maret 2020, hlm. 919-928 http://j-ptiik.ub.ac.id

Implementasi *Load Balancing Web Server* dengan Algoritma *Source IP Hash* pada *Software Defined Network* (SDN)

Anita Sumiati[1], Primantara Hari Trisnawan[2], Mochammad Ali Fauzi[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]anitasumiati2@gmail.com, [2]prima@ub.ac.id, [3]moch.ali.fauzi@ub.ac.id

Abstrak

Software Defined Network (SDN) adalah arsitektur jaringan yang memisahkan *control plane* dan *data plane*. Arsitektur jaringan SDN membagi permintaan yang masuk untuk diberikan ke suatu server. SDN menggunakan *OpenFlow versi* 1.3 sebagai protokol dan *load balancing* untuk mendistribusikan beban pada dua atau bahkan lebih koneksi jaringan secara seimbang. *Load balancing* memiliki algoritma yang digunakan untuk membagikan *request* ke setiap server. Terdapat beberapa algoritma yang telah digunakan sebelumnya, salah satunya adalah algoritma *least traffic*. Penelitian ini menggunakan algoritma *Source IP Hash*, dan *Ryu* sebagai *controller* yang berfungsi untuk mengalokasikan server dengan melakukan *hash* pada alamat IP sumber. Algoritma ini akan diuji dan dianalisis kinerjanya pada, *response time*, *throughput*, dan *packet loss*. Nilai *throughput* dari algoritma *source ip hash* mencapai 206 mbps sedangkan algoritma *least traffic* 106 mbps. *Packet Loss* yang terjadi saat melakukan pengiriman data pada algoritma *source IP hash* mencapai 0% sedangkan algoritma *least traffic* mengalami *packet loss* dengan kategori *high* mencapai 4,009%. Dari kedua hasil pengujian *troughput* dan *packet loss* disimpulkan bahwa algoritma *source IP hash menghandle request* dari *client* lebih baik dibanding *least traffic*. Algoritma *Source IP Hash* dapat mengalokasikan alamat IP sumber yang sama masuk ke server yang sama, sedangkan algoritma *least traffic* harus melakukan pemilihan server dengan melakukan perulangan berdasarkan *traffic* rendah.

Kata kunci: *load balancing, software defined network, OpenFlow, ryu controller, source IP hash*

Abstract

Software Defined Networking (SDN) is a network architecture that separates the control plane from the data plane. SDN network architecture divides incoming requests to be given to a server. SDN uses OpenFlow version 1.3 as a protocol and load balancing to distribute the load on two or even more balanced network connections. Load balancing has an algorithm that is use to distribute requests to each server. There were several algorithms that have been used before, one of them is the least traffic algorithm. This research uses the Source IP Hash algorithm, and Ryu as the controller that functions to allocate the server by doing a hash on the source IP address. This algorithm is tested and analyzed its performance on, response time, throughput, and packet loss. The throughput value of the source IP hash algorithm reaches 206 mbps while the least traffic algorithm is 106 mbps. Packet Loss that occured in the source IP hash algorithm is 0% while the least traffic algorithm experienced packet loss with the category high as much as 4.009%. From the two results of throughput and packet loss tested, it is concluded that the source IP hash algorithm handles requests from clients better than least traffic. The Source IP Hash algorithm could allocated the same source IP address to the same server, while the least traffic algorithm must selected the server by making a loop based on low traffic.

Keywords: *load balancing, software defined network, OpenFlow, ryu controller, source IP hash*

1. PENDAHULUAN

Penggunaan internet dan lalu lintas internet saat ini telah meningkat pesat. Jaringan harus menangani banyak klien dengan server tunggal. *Software Defined Network* (SDN) memungkinkan untuk meningkatkan kinerja jaringan/server. SDN adalah teknologi jaringan baru yang sedang berkembang yang dapat meningkatkan layanan jaringan. Perangkat lunak

mendefinisikan penyeimbangan beban berbasis jaringan digunakan untuk mengelola beban lalu lintas yang sangat besar yang mendistribusikan beban diantara berbagai server (Ikram, Arif, Ayub, & Arif, 2018).

Pendistribusian beban *traffic* pada dua jalur atau lebih koneksi secara seimbang disebut dengan *Load Balancing*. Pendistribusian beban yang seimbang dapat mengoptimalkan *traffic*, meningkatkan *throughput*, memperkecil nilai *response time*, dan mencegah terjadinya kelebihan beban pada salah satu jalur koneksi (Sirajuddin, Affandi, & Setijadi, 2012). Apabila salah satu server tidak mampu menerima *traffic* dari pengguna maka, *load balancer* memutuskan server mana yang dapat menghandlle *traffic* dimana server dapat berupa fisik atau *virtual*. *Load balancer* membantu server memindahkan data secara efisien, mengoptimalkan penggunaan sumber daya, dan mencegah kelebihan beban pada server. Adapun beberapa algoritma yang dapat digunakan saat ini dalam *load balancing* yaitu *Round Robin*, *Least Connection*, *Least Response Time*, *IP Hash* dan lain-lain (Networks, 2019).

SDN menyediakan kontrol yang *fleksibel*, *load balancing* di SDN memisahkan bidang kontrol jaringan fisik dari bidang data. Penyeimbang beban berbasis SDN memungkinkan kontrol beberapa perangkat. Kontrol jaringan dapat diprogram secara langsung untuk layanan aplikasi yang lebih *responsif* dan *efisien* sehingga jaringan dapat menjadi lebih cepat. SDN *load Balancing* adalah *load balancing* jaringan yang ditentukan perangkat lunak. Sebuah penyeimbang beban berbasis SDN secara fisik memisahkan *control plane* dari *data plane*. Lebih dari satu perangkat dapat dikontrol pada saat bersamaan ketika *load balancing* menggunakan SDN, penyeimbangan beban menjadi lebih optimal. (Networks, 2020).

Berdasarkan penjelasan di atas, Kebijakan IP *Hash* menggunakan alamat IP sumber permintaan yang masuk sebagai kunci *hashing* untuk merutekan lalu lintas ke server *backend* yang sama (Cloud, 2020). Algoritma *source IP hash* bersifat *deterministik*, yang mana setiap langkah dalam algoritma hanya dapat di proses satu kali. Apabila jalur tidak ditemukan maka algoritma dianggap selesai. Algoritma optimasi ini akan selalu menghasilkan solusi yang sama untuk setiap input yang diberikan. Fungsi *hash* diterapkan pada alamat IP sumber dari permintaan yang masuk dan *hash* harus memperhitungkan jumlah server dan bobot masing-masing server (Assmann, 2013).

2. LANDASAN KEPUSTAKAAN

Landasan Kepustakaan menjelaskan berbagai teori dan kajian pustaka yang mendukung penelitian Implementasi Algoritma *Load Balancing Web Server* berbasis SDN.

2.1 Kajian Pustaka

Kajian Pustaka berisi tentang berbagai penelitian yang telah dilakukan sebelumnya yang berkaitan dengan Algoritma *Load Balancing* pada *Software Defined Network* yang diuraikan pada Tabel 1.

Tabel 1 Kajian Pustaka

No	Judul, (Nama Penulis, Tahun)	Persamaan	Perbedaan	
			Penelitian Sebelumnya	Rencana Penelitian
1	"Implementasi *Load Balancing Web Server* Dengan *Algoritme Weighted Least Connection* Pada *Software Defined Network*", (Kharim, 2018)	Menjalankan *load balancing web server* pada arsitektur jaringan SDN	Melakukan implementasi menggunakan *algoritme weighted least connection* dengan *controller POX*	Mengimplementasikan sistem *load balancing web server* menggunakan algoritma *Source IP Hash* berbasis SDN menggunakan *controller RYU*
2	"Implementasi *Load Balancing* Pada Server Dengan Menggunakan *Algoritme Least Traffic* Pada *Software Defined Network*", (Fauzi, 2018)	Mengimplementasikan *Ryu controller* pada algoritma *load balancing web server* berbasis SDN	Menjelaskan tentang *algoritme least traffic* pada SDN dengan *Ryu controller*	Mengimplementasikan algoritma *load balancing web server* menggunakan algoritma *Source IP Hash* berbasis SDN
3	"Implementasi *Algoritme Weighted Least Connection* Berbasis *Agen* pada *POX Controller* Untuk *Load Balancing Web Server* Pada *Software Defined Network*", (Sumbayak, 2019)	Menjalankan algortima *Load Balancing Web Server* pada jaringan SDN	Menjelaskan tentang *algoritme Weighted Least Connection* menggunkan *POX Controller* pada SDN Berbasis *Agen*	Mengimpelentasikan sistem *load balancing web server* menggunakan algoritma *Source IP Hash* pada SDN

2.2 *Load Balancing*

Load Balancing merupakan suatu perangkat yang mendistribusikan *network traffic* atau aplikasi ketika melintasi sekelompok server. *Load Balancing* digunakan ketika sebuah server memiliki jumlah *user* yang melebihi kapasitas dengan teknik mendistribusikan beban *traffic* pada dua atau lebih jalur koneksi secara seimbang. *Traffic* pada jaringan dapat berjalan optimal, memaksimalkan *throughput*, memperkecil *response time* dan menghindari *overload* pada salah satu jalur koneksi.

2.3 *Source IP Hash*

Source IP Hash adalah algoritma *load balancing* yang mengambil *source* IP dari *client* untuk menghasilkan kunci *hash* yang unik. Kunci ini digunakan untuk mengalokasikan

client ke server tertentu. Metode penyeimbangan beban ini dapat memastikan bahwa *client* diarahkan ke server yang sama dengan yang digunakan sebelumnya. Saat melakukan pengiriman *Source IP Hash*, fungsi *Hash* diterapkan pada alamat IP sumber dari permintaan yang masuk. Hal tersebut memastikan bahwa ketika suatu koneksi melewati perangkat, selama itu menggunakan alamat IP sumber yang sama, koneksi akan tetap *persisten* (Systems, 2019).

2.4 *Software Defined Network* (SDN)

Software Defined Network (SDN) adalah arsitektur jaringan yang bersifat dinamis, mudah dikelola, hemat biaya, dan mudah beradaptasi hingga sebuah sistem menjadi ideal untuk *bandwidth* tinggi (Fundation, 2012). *Data plane* dan *control plane* pada SDN bertujuan untuk mendefinisikan suatu perangkat *switch/router* yang terdapat pada *data plane* secara sederhana. *Software Defined Network* (SDN) memiliki *interface standard* untuk memprogram perangkat pada jaringan. Pemisahan secara *fisik/eksplisit* antara *forwading/data plane* dan *control plane* merupakan aspek penting dari *Software Defined Network.*

3. METODOLOGI

Metodologi menjelaskan tentang studi literatur, analisis kebutuhan, perancangan sistem, pengujian dan analisis sistem, dan kesimpulan pada penelitian 'Implementasi *Load Balancing Web Server* dengan Algoritma *Source IP Hash* pada *Software Defined Network'*.

3.1 Studi Literatur

Studi literatur menjelaskan berdasarkan beberapa teori yang berhubungan dengan beberapa konsep, seperti *load balancing, software defined network, web server* dan *OpenFlow* yang digunakan dalam penelitian ini. Teori-teori tersebut diperoleh dari berbagai sumber buku, *paper* (jurnal ilmiah), dan dokumen dari *internet* yang berkaitan dengan penelitian.

3.2 Analisis Kebutuhan

Analisis kebutuhan sistem bertujuan untuk menganalisa kebutuhan yang diperlukan pada penelitian ini. Kebutuhan ini mencakup kebutuhan utama (*openflow, ryu controller,* SDN, dan *mininet*) dan kebutuhan pendukung (*Ubuntu server* versi 19.04). *Algoritma load balancing web server* pada SDN dibangun untuk mengimplementasikan sistem, dengan menjalankan sistem pada jaringan berbasis SDN. Paket HTTP *request* yang dikirimkan dari beberapa *client* dapat dilakukan dengan satu alamat *destination* yang sama.

Aplikasi *wireshark* pada penelitian ini juga dibutuhkan, untuk mendukung berjalannya penelitian ini. *Wireshark* digunakan untuk untuk melihat trafik pada saat menerapkan algoritma *Source IP Hash* pada jaringan SDN. Penelitian ini menggunakan bahasa pemograman *python* yang digunakan untuk mengimpelmentasikan algoritma *Source IP Hash* di *controller ryu*.

3.3 Perancangan Sistem

Perancangan sistem pada penelitian ini dilaksanakan setelah seluruh kebutuhan sistem dipenuhi dengan tujuan pembangunan sistem lebih terarah dan terstruktur. Sistem nantinya akan dipecah menjadi beberapa *subsistem* yaitu *controller*, *switch*, *web server*, dan *client*. *OpenSwitch* akan diimplementasikan sebagai SDN *switch* pada *emulator Mininet* dengan *Operating Sistem ubuntu 19.04 LTS*. *Ubuntu 19.04* dijalankan pada *VirtualBox* yang ada di *OS Windows 10 64-bit*.

3.4 Implementasi Sistem

Implementasi sistem dilakukan setelah perancangan sistem dan perancangan algoritma selesai dilakukan. Pertama melakukan *instalasi Mininet* dan *Ryu controller* pada *sistem operasi linux,* kemudian membangun arsitektur *Load Balancing web server* pada *Software Defined Network* di *Mininet* server, lalu membuat algoritma *Source IP Hash* sebagai mekanisme *Load Balancing*, selanjutnya melakukan akses ke halaman *web server*, dan yang terakhir implementasi pengujian dengan parameter *Response Time, Throughput, Packet Loss dan Cek Traffic* untuk mengetahui performa dan kinerja *web server*.

3.5 Pengujian dan Analisis Sistem

Pengujian dan analisis sistem pada penelitian ini dilakukan setelah mengetahui kinerja dari suatu sistem sesuai dengan kebutuhan atau tidak. Setelah itu melakukan analisis pengujian berdasarkan hasil dari *Response Time, Throughput, Packet Loss,* dan *Cek Traffic.*

4. PERANCANGAN DAN IMPLEMENTASI

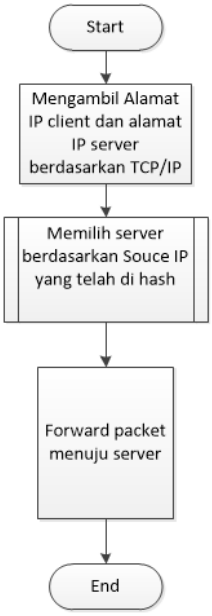
Perancangan dan implementasi, menjelaskan tahap-tahap dalam perancangan dan implementasi sistem *Load Balancing Source IP Hash* pada *Software Defined Network* sesuai dengan yang dibahas pada metodologi penelitian.

4.1 Perancangan Sistem

Bagian ini, menjelaskan sesuai perancangan yang akan dibangun pada sistem berdasarkan kebutuhan sistem yang telah dispesifikasikan.

4.1.1 Diagram Alir Perancangan Sistem

Perancangan sistem pada penelitian ini terdiri atas beberapa tahapan yang harus dilakukan. Tahapan-tahapan tersebut digambarkan pada Gambar 1.

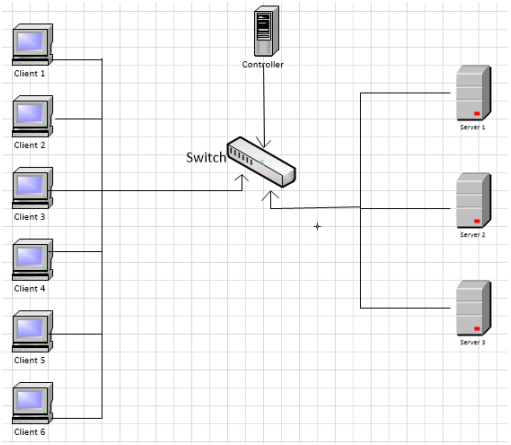


Gambar 1 *Flowchart* Keseluruhan Sistem

Gambar 1 adalah *flowchart* keseluruhan sistem. Sistem dimulai dari pengambilan alamat IP *client* dan alamat IP server, kedua algoritma *Source IP Hash* akan menentukan jalur menuju server berdasarkan IP yang telah di *hash* dan selanjutnya *switch* akan meneruskan paket menuju server yang telah ditentukan.

4.1.2 Perancangan Arsitektur *Software Defined Network*

Perancangan dilakukan dengan membangun sebuah arsitektur SDN minimal 1 *controller* dan 1 *switch* untuk mendukung *protocol OpenFlow*. Penelitian terdiri dari 4 *subsistem* (*controller*, *switch*, *client*, dan server) sesuai dengan gambaran umum sistem yang telah dijelaskan pada analisis kebutuhan.



Gambar 2 Topologi Perancangan Sistem

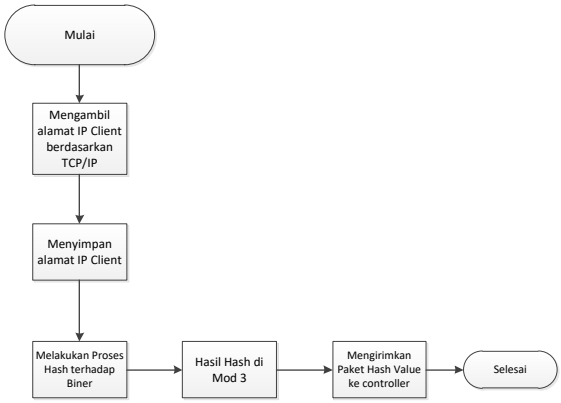
Gambar 2 adalah perancangan sistem yang akan digunakan pada penelitian ini. Sistem ini dibagi menjadi beberapa *subsistem*, yaitu 1 *controller*, 1 *switch*, *3 web server*, dan *6 client*.

4.1.3 Perancangan Sistem *Load Balancing*

Perancangan *load balancing system* dilakukan dengan membangun sebuah fungsi *load balancing server* dalam membagi beban server berdasarkan hasil *hash* pada IP *client.*

4.1.4 Perancangan Algoritma *Source IP Hash*

Metode *hashing* dapat diterapkan ke alamat IPv4 dengan panjang 32 bit. Mempertimbangkan skenario dimana tiga layanan (Server 1, Server 2, dan Server 3) terikat ke server *virtual*, metode *hash* dihitung dengan *md5*, dan nilai *hash* didapatkan dari hasil *hash* md5 dari biner IP *client* kemudian di *mod* 3. Gambar 3 merupakan diagram yang menggambarkan proses ini.



Gambar 3 Diagram Alir Proses *Hassing*

4.2 Implementasi Sistem

Penelitian ini mengimplementasikan *Software Defined Network* (SDN) pada *emulator Mininet*. *Switch OpenFlow* dan *Controller* didukung oleh *emulator mininet*. *Library Python* harus sudah terinstal terlebih dahulu sebelum penginstallan *mininet* dilakukan.

4.2.1 Mininet

Mininet merupakan *emulator* jaringan pada *SDN* berbasis CLI. *Mininet* digunakan untuk membuat topologi. Dengan *mininet emulator* penelitian dapat mempelajari SDN tanpa harus menggunakan perangkat yang aslinya. Dengan mengunakan *mininet emulator* sistem SDN berjalan seperti dunia nyata.

4.2.2 Ryu

Ryu adalah sebuah *controller* dalam SDN. Ryu dirancang agar meningkatkan kemampuan dalam jaringan yang bermanfaat untuk mepermudah dan mengatur. Secara umum *controller* merupakan otak dari *Software Defined Network*.

4.2.3 Implementasi Aplikasi Server *Load Balancing*

Implementasi aplikasi *load balancing server* berbasis SDN dilakukan berdasarkan rancangan yang telah dibuat sebelumnya. Terdapat 4 implementasi yang harus dilakukaan pada proses ini, yaitu impelentasi topologi *emulator Mininet*, konfigurasi IP, implementasi *web server*, implementasi *client*, dan implementasi *ryu*.

4.2.4 Pengembangan Program Controller

Program *controller* menggunakan *Python* Untuk menyeimbangkan beban server ditentukan berdasarkan hasil *hash* dari alamat IP *client*. Berikut ini merupakan *source code* algoritma *Source IP Hash*.

Tabel 2 *Source Code* Algoritma *Source* IP *Hash*

No	Source Code Algoritma Source IP Hash
1	if tcp_header.dst_port == 80:
2	source_ip = ip_header.src
3	arr_oktet = str(source_ip).split('.')
4	
5	biner1 = bin(int(arr_oktet[0]))
6	[2:].zfill(8)
7	biner2 = bin(int(arr_oktet[1]))
8	[2:].zfill(8)
9	biner3 = bin(int(arr_oktet[2]))
10	[2:].zfill(8)
11	biner4 = bin(int(arr_oktet[3]))
12	[2:].zfill(8)
13	biner_source_ip = (biner1) + str('.') +
14	(biner2) + str('.')
15	+ (biner3) + str('.') + (biner4)
16	
17	# Hashing
18	hashing = md5(biner_source_ip)
19	mod = hashing % 3
20	index = mod
21	
22	server_mac_selected =
23	self.serverlist[index]['mac']
24	server_ip_selected =
25	self.serverlist[index]['ip']
26	server_outport_selected =
27	int(self.serverlist[index]['outport'])
28	index = index + 1
29	
30	print("biner", biner_source_ip)
31	print("hash", hashing)
32	print("mac", server_mac_selected)
33	print("ip", server_ip_selected)
34	print("outport", server_outport_selected)
35	
36	print("Server ", server_ip_selected)

Tabel 2 adalah *source code* dari algoritma *Source IP Hash* yang digunakan untuk mengalokasikan *client* ke server tertentu.

5. PENGUJIAN DAN ANALISIS

Pengujian dan analisis menjelaskan tentang, hasil pengujian yang dilakukan dan menilai kebutuhan yang sudah dispesifikasikan pada bagian sebelumnya**.** Analisis dilakukan untuk menilai suatu sistem yang diharapkan dan sebagai acuan dalam menarik kesimpulan dari penelitian ini.

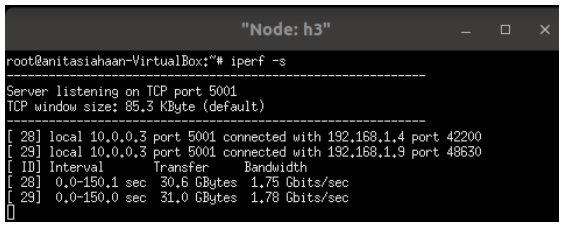
5.1 Pengujian

5.1.1 Pengujian *Throughput*

Tabel 3 *Syntax* Pengujian *Throughput*

Host	***Syntax***
Server	Iperf –s
Client	Iperf –c virtual_IP –t waktu_kirim –i interval_waktu

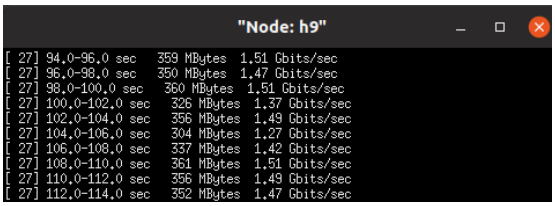
Tabel 3 adalah *syntax* yang digunakan melakukan uji parameter *throughput*.



Gambar 4 Pengujian *Throughput* pada Server

Gambar 4 merupakan hasil pengujian

throughput pada sisi server dengan menggunakan h3. Protokol yang digunakan pada pengujian ini adalah TCP. Setelah proses *iperf* sisi *client* selesai, maka hasil akan ditampilkan pada tabel yang terminal server.



Gambar 5 Pengujian *Throghput* pada Server

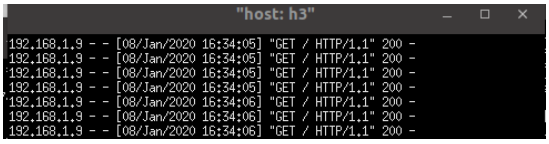
Gambar 5 adalah pengujian *throughput* pada host h9 yang berfungsi sebagai *client*. Hasil pada kolom *bandwidth* adalah rata-rata *bandwidth* pada bagian bawah yang diperoleh dengan percobaan selama 150 detik sedangkan pada bagian *transfer* merupakan jumlah paket yang berhasil dikirim selama waktu 150 detik.

5.1.2 Pengujian *Response Time*

Tabel 4 *Syntax* Pengujian *Response Time*

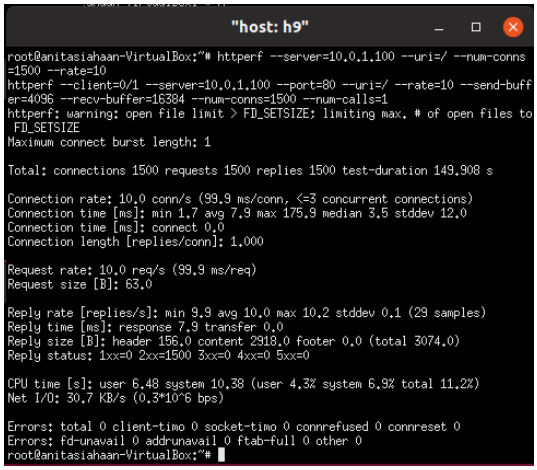
Host	**Syntax**
Server	*python –m SimpleHTTPServer 80*
Client	*httperf –server=virtual_IP –uri=/ --num-conns=jumlah_request –rate=waktu_request*

Tabel 4 adalah *syntax* yang dijalankan untuk mengetahui lama waktu yang terjadi saat sistem melakukan pengiriman paket.



Gambar 6 Pengujian *Response Time* pada Server

Gambar 6 adalah proses saat menjalankan pengujian *response time* pada *host* server h1, h2 dan h3 yang berfungsi sebagai server. Setelah proses pengiriman selesai maka hasil dari pengiriman akan ditampilkan di *terminal* server.



Gambar 7 Pengujian *Response Time* pada *Client*

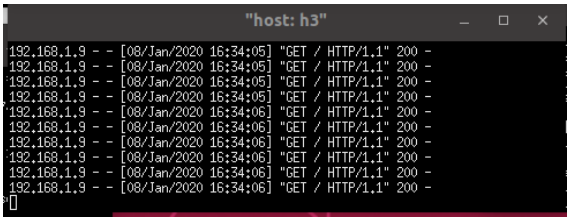
Gambar 7 adalah proses pengujian *Response Time* pada host yang berfungsi sebagai *client*. *Host* yang berfungsi sebagai *client* harus menuliskan *httperf* dan alamat *virtual* IP pada *host* yang dijalankan sebagai *client*.

5.1.3 Pengujian *Packet Loss*

Tabel 5 Pengujian *Packet Loss*

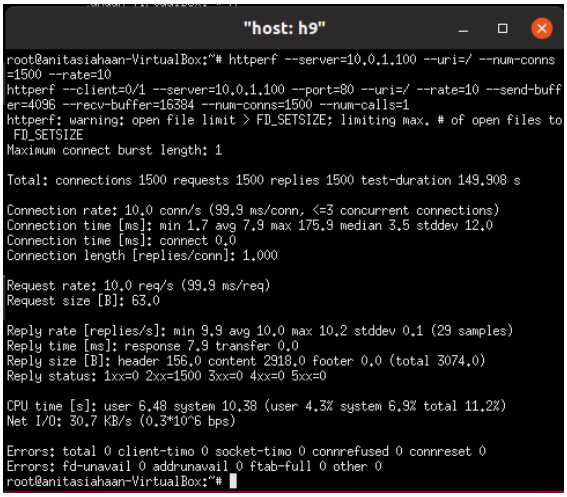
Host	**Syntax**
Server	*python –m SimpleHTTPServer 80*
Client	*httperf –server=virtual_IP –uri=/ --num-conns=jumlah_request –rate=waktu_request*

Tabel 5 adalah *syntax* saat menjalankan pengujian *packet loss* dengan melakukan *ping* kesetiap server.



Gambar 8 Pengujian *Packet Loss* Server

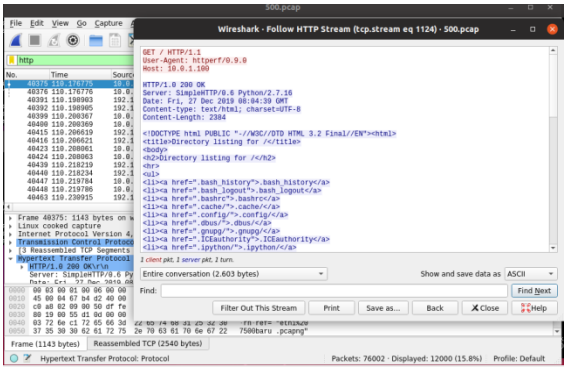
Gambar 8 adalah hasil saat *request* diterima oleh server dengan menjalankan *python –m SimpleHTTPServer 80* seperti pada tabel diatas. Perintah dijalankan pada *host* h1, h2, dan h3 yang berfungsi sebagai server. Setelah proses pengiriman selesai maka hasil dari *request* akan ditampilkan di *terminal client*.



Gambar 9 Pengijian *Packet Loss* sisi *Client*

Gambar 9 adalah hasil dari pengujian *Response Time* pada host yang berfungsi sebagai *client*. *Host* yang berfungsi sebagai *client* dengan menjalankan *tools httperf* dan menuliskan alamat server dengan alamat *virtual* IP pada *host* yang akan dijalankan sebagai *client*. Setelah proses selesai hasil pengiriman ditampilkan pada *host client* seperti pada gambar di atas.

5.1.4 Pengujian *Cek Traffic*

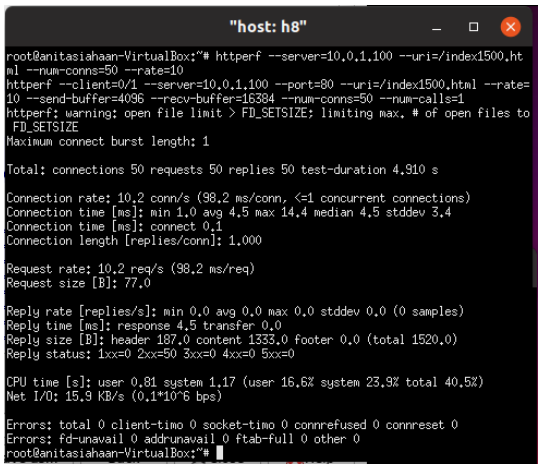


Gambar 10 Pengujian Cek *Traffic*

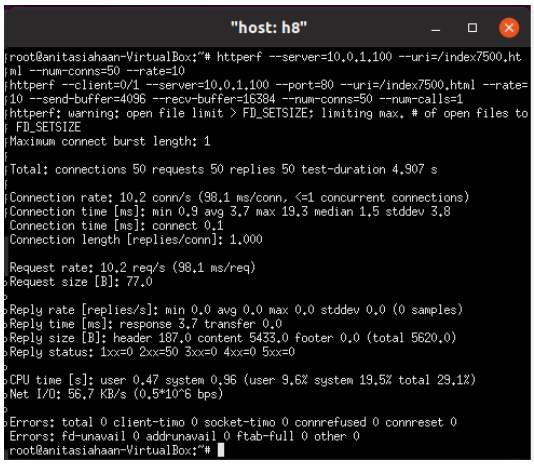
Gambar 10 menampilkan hasil *capture* menggubakan aplikasi *wireshark* berupa informasi *time*, *source*, *destination*, *protocol*, *length* dan informasi dari paket yang didapatkan berdasarkan hasil *capture* saat melakukan pengujian. Berdasarkan gambar di atas *request* paket pada *client* dapat diterima oleh server dan dapat ditangkap/ *capture* menggunakan *wireshark*.

5.1.5 Pengujian Web Server

Gambar 11 adalah hasil salah satu pengujian dengan mengirimkan total *request* 50 dengan data 1520 byte. Arti dari perintah di atas yaitu mengirim index1500.html dengan jumlah *request* 50 dan rate 10 *request/second* dengan alamat *destination* IP *load balancing* dengan menggunakan *tools httperf*. *Httperf* melakukan pengiriman *request* dari *client* ke server, setelah menjalankan perintah tersebut didapatkan nilai *throughput*, *response time* dan *packet loss*. Pada gambar di atas, dilakukan dengan perintah dari *client* menuju server secara bersamaan.

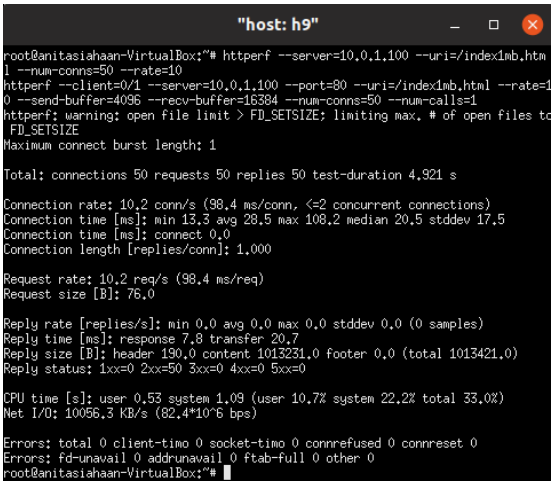


Gambar 11 Pengujian *Web Server* dengan Data Kecil



Gambar 12 Pengujian *Web Server* dengan Data Besar

Gambar 12 adalah hasil dari pengujian dengan mengirimkan *request* sebnayak 50 dengan *size* data yang dikirim sebesar 5620 byte. Arti dari perintah di atas yaitu, *client* mengirimkan *packet* data *index7500.html* dengan jumlah *request* 50 dan rate 10 *request/second* ke alamat IP *load balancing* sebagai alamat IP server menggunakan *httperf*. Pada gambar di atas *client* mengirimkan *request packet* dari *client* ke server.

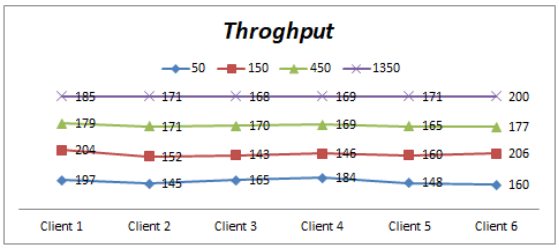


Gambar 13 Pengujian *Web Server* dengan Data 1 MB

Gambar 13 adalah hasil dari pengujian dengan mengirimkan *request* sebnayak 50 dengan *size* data yang dikirim sebesar 1013421 byte. Arti dari perintah di atas yaitu, *client* mengirimkan *packet* data *index1mb.html* dengan jumlah *request* 50 dan rate 10 *request/second* ke alamat IP *load balancing* sebagai alamat IP server menggunakan *httperf*. Pada gambar di atas *client* mengirimkan *request packet* dari *client* ke server.

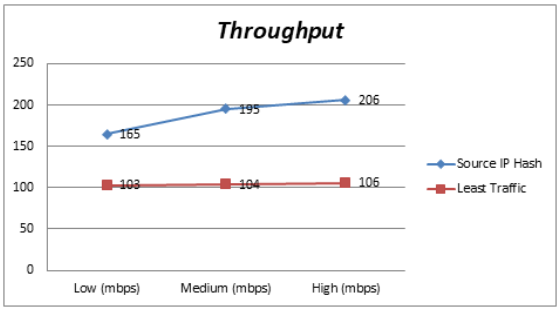
5.2 Analisis Hasil Pengujian

5.2.1 Pengujian *Throughput*



Gambar 14 Hasil Pengujian *Throughput Source IP Hash*

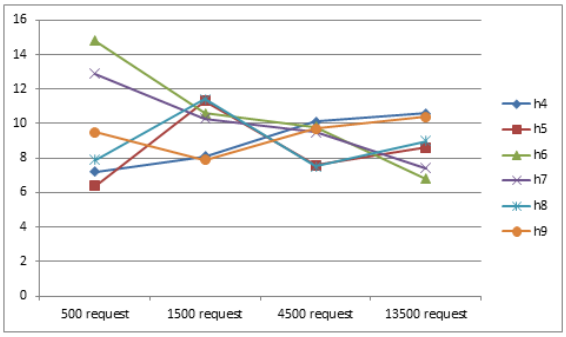
Gambar 14 adalah grafik dari hasil pengujian *throughput* yang ditentukan dengan 3 *host* sebagai server dan 6 *host* sebagai client. Nilai *throughput* pada pengujain dengan interval waktu 2 detik dan waktu kirim 50 detik, 150 detik, 450 detik dan 1350 detik menghasilkan nilai paling rendah 143 Mbps, menengah 177 Mbps, dan tertinggi 206 Mbps yang tercetak pada terminal *host* client.



Gambar 15 Perbandingan *Throughput*

Gambar 15 adalah perbandingan antara algoritma *source IP hash* dan *least traffic* dengan menggunakan interval waktu 2 detik dan waktu kirim 10 detik. Grafik di atas merupakan nilai *throughput* dengan 3 host sebagai server. *low* 103 Mbps, *medium* 104 Mbps dan *high* 106 Mbps sedangkan algoritma *source IP hash* dengan kategori *low* 165 Mbps, *medium* 195 Mbps, dan *high* mencapai 206 Mbps.

5.2.2 Pengujian *Response Time*



Gambar 16 Hasil Pengujian *Response Time*

Gambar 16 adalah hasil dari melakukan pengujian dengan parameter *response time*. Pengujian digunakan untuk mendapatkan data waktu respon server ke *client* saat melakukan *request*. Dari gambar grafik di atas dapat disimpulkan bahwa *response time* terendah 6,4 terdapat pada host H5 dengan *request* sebanyak 500 dan *rate* 10 *request/second*. *Response time* tertinggi 14,8 terdapat pada host H6 dengan *request* sebanyak 500 dan *rate* 10 *request/second*.

5.2.3 Pengujian *Packet Loss*

Tabel 3 Hasil Pengujian *Packet Loss Web Server*

	Parameter Pengujian	
	Source IP Hash	*Least Traffic*
Low	*0%*	*2,97*
Medium	*0%*	*3,403%*

High	*0%*	*4,009%*

Tabel 3 adalah hasil yang didapatkan dari pengujian *packet loss* menggunakan *tools httperf* dari *client* ke server. Dari hasil pengujian *packet loss* didapatkan 0% *packet loss* yang terjadi saat melakukan *request* semua *client* ke server. Dari tabel diatas algoritma *source IP hash* mengalami *packet loss* 0% berbeda dengan algoritma *Least Traffic* mengalami *packet loss* 4,009%.

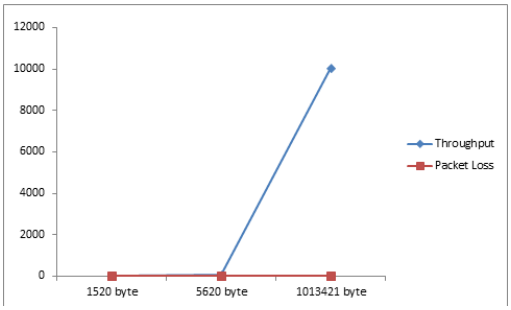
5.2.4 Pengujian *Cek Traffic*

Tabel 4 Hasil Pengujian *Traffic*

No	Server yang diuji	Banyak *Request*	*Source IP Hash*
1	Server 1	2	192.168.1.5 dan 192.168.1.8
2	Server 2	2	192.168.1.6 dan 192.168.1.7
3	Server 3	2	192.168.1.4 dan 192.168.1.9

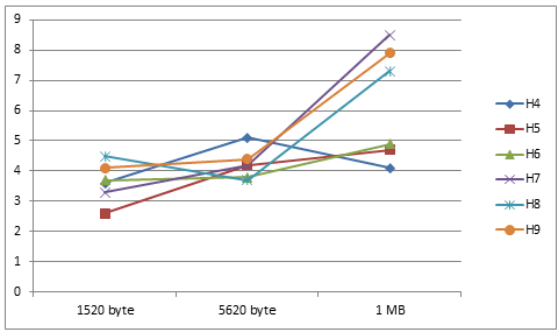
Tabel 4 adalah hasil saat menjalankan algoritma *Source IP Hash* dengan jumlah *client* 6. Berdasarkan hasil saat menjalankan *Algoritma Source IP Hash* server 1, server 2, dan server 3 menerima *request* yang sama dari *client*.

5.3 Pengujian Web Server



Gambar 17 Hasil Pengujian *Web Server*

Gambar 17 adalah hasil saat menjalankan algoritma *Source IP Hash* dengan mengirimkan *request* sebanyak 50 dengan size data yang berbeda-beda. Pengujian *webserver* menggunakan *httperf* pada *host client*. Paket yang dikirim sebesar 1520 *byte* kategori *low*, 5620 *byte* kategori *high* dan 1MB. Dari hasil pengujian didapatkan rata-rata *throughput* dengan kategori *request* data besar 56,7 KB/s, untuk kategori *request* data kecil 15,9 KB/s, dan data sebesar 1MB menghasilkan nilai *throughput* 10041,23 KB/s. Pengujian dengan melakukan *request* sebanyak 50 *request* dengan rate 10 *request/second* menghasilkan nilai rata-rata *packet loss* 0% untuk data besar, 0% untuk data kecil dan 0% untuk data 1MB.



Gambar 18 *Response Time* Pengujian *Web Server*

Gambar 18 adalah hasil saat melakukan pengujian untuk mendapatkan data waktu respon server ke *client* saat melakukan *request*. Dari gambar grafik di atas dapat disimpulkan bahwa *response time* terendah terdapat pada host H5 *request* sebesar 1520 *byte* dengan jumlah *request 50* dan *rate* 10 *request/second*. R*esponse time* tertinggi pada host H7 dengan *request* sebesar 1 MB dengan jumlah *request 50* dan *rate 10 request/second.* Nilai rata-rata yang didapat dari masing-masing pengujian adalah 3,63 untuk 1520 *byte*, 4,23 untuk 5620 *byte* dan 6,23 untuk 1 MB.

6. KESIMPULAN DAN SARAN

6.1 Kesimpulan

Penelitian dengan judul "Implementasi *Load Balancing Web Server* dengan Algoritma *Source IP Hash* pada *Software Defined Network*" dapat disimpulkan bahwa:

1. *Source IP Hash* diimplementasikan dengan menerima *request* dari *client* yang masuk pada jaringan. Penelitian ini menggunakan bahasa *python* pada *ryu controller* dengan memanfaatkan arsitektur jaringan *Software Defined Network* (SDN). Pengujian ini menunjukan bahwa setiap *Source* IP yang masuk pada proses *controller* akan dilakukan *hash* terhadap IP.
2. Nilai *throughput* tertinggi dari algoritma *Source IP Hash* mencapai 206 Mbps sedangkan pada algoritma *least traffic* mencapai 106 Mbps. Nilai rata-rata *response time* tertinggi server ke *client* saat melakukan *request* adalah 14,8 pada host h6 dengan jumlah *request* 500, sementara nilai respon terendah terdapat pada host h5 dengan nilai 6,4 dengan jumlah *request*

500. *Packet loss* yang didapat saat melakukan pengiriman paket sebanyak 500, 1500, 4500 dan 1350 *request* adalah 0% sedangkan algoritma *Least Traffic* mengalami *packet loss* 4,009%. Hasil pengujian dengan pengujian *iperf* didapatkan ketika semakin banyak waktu yang dibutuhkan dalam melakukan pengiriman, maka semakin besar nilai *throughput* yang dihasilkan. Waktu yang dilakukan dalam pengujian selama 50, 150, 450 dan 1350 detik dengan *interval* waktu 2 detik. Pada pengujian *web server* menggunakan *httperf* menghasilkan nilai rata-rata *throughput* 56,7 Kb/s saat mengirimkan data besar 5620 *byte* sedangkan data kecil 1520 *byte* menghasilkan rata-rata *throughput* 15,9 Kb/s dan data 1 MB menghasilkan rata-rata *throughput* 10041,23 Kb/s, *packet loss* yang terjadi 0% dan menghasilkan nilai rata-rata *response time* 3,63, 4,23 dan 6,23.

6.2 Saran

Berdasarkan hasil dari pengujian sistem, dapat diberikan saran-saran untuk pengembangan selanjutnya, antara lain:

1. Melakukan implementasi sistem *load balancing* dengan jumlah *request* yang lebih banyak lagi dan waktu yang lebih lama.
2. Pada penelitian selanjutnya dapat menambah parameter pengujian yang lain agar dapat diketahui bagaimana kinerja algoritma dan dapat dianalisis lebih lanjut.

7. DAFTAR PUSTAKA

Assmann, B. (2013, April 22). *Client IP Persistence or Source IP Hash Load Balancing*. Retrieved Maret 11, 2020, from Haproxy: https://www.haproxy.com/blog/client-ip-persistence-or-source-ip-hash-load-balancing/

Cloud, O. (2020). *How Load Balancing Policies Work*. Retrieved Maret 2020, from https://docs.cloud.oracle.com/en-us/iaas/Content/Balance/Reference/lbpolicies.htm#hash

Sirajuddin, S., Affandi, A., & Setijadi, E. (2012). Rancang Bangun Server Learning Management System Menggunakan Load Balancer dan Reverse Proxy.

Ikram, A., Arif, S., Ayub, N., & Arif, W. (2018). *Load Balancing In Software Defined Networking (SDN)*. Retrieved Maret 14, 2020, from Semantic Scholar: https://www.semanticscholar.org/paper/Load-Balancing-In-Software-Defined-Networking-(SDN)-Ikram-Arif/e9e21a0fe67cbb77b4eb84bbc62111c58570f272

Networks, A. (2019). *What Is Load Balancing*. Retrieved Desember 2019, from AVI Networks Now part of VMware: https://avinetworks.com/what-is-load-balancing/

Networks, A. (2020). *SDN Load Balancing*. Retrieved Maret 20, 2020, from https://avinetworks.com/glossary/sdn-load-balancing/

Systems, C. (2019). *Citrix Product Documentation*. Retrieved Januari 6, 2019, from https://docs.citrix.com/en-us/netscaler/12/load-balancing/load-balancing-customizing-algorithms/hashing-methods.html

Fundation, O. N. (2012). *ONF Connect*. Retrieved Februari 2019, from https://www.opennetworking.org/sdn-definition/?nab=0

Fauzi, N. (2018). Implementasi Load Balancing Pada Server Dengan Menggunakan Algoritme Least Traffic Pada Software Defined Network.

Kharim, H. (2018). Implementasi Load Balancing Web Server Least Connection Pada Software Defined Network.

Sumbayak, G. S. (2019). Implementasi Algoritme Weighted Least Connection Berbasis Agen pada POX Controller Untuk Load Balancing Web Server Pada Software Defined Network.