Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 1, Januari 2019, hlm. 133-139 http://j-ptiik.ub.ac.id

Implementasi *Server Failover* Pada *Software Defined Network*

Dea Asmara Gita[1], Sabriansyah Rizqika Akbar [2], Widhi Yahya[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]deaa.asmara@gmail.com, [2]sabrian@ub.ac.id, [3]widhi.yahya@ub.ac.id

Abstrak

Arsitektur *Client-Server* adalah sebuah arsitektur pada jaringan komputer yang melibatkan pertukaran data antar dua perangkat yaitu *client* dan *server*. Peranan *server* pada arsitektur *Client-Server* sangatlah penting, karena jika *server* mengalami *down* maka client tidak dapat mengakses layanan yang disediakan oleh *server*. Untuk mengatasi hal tersebut diperlukan sebuah sistem yang memiliki sifat *high availability*. Beberapa sistem *high availability* yang pernah diterapkan menggunakan model *master* dan *slave*. Pada penelitian ini arsitektur *Client-Server* yang bersifat *high availability* diwujudkan dengan menggunakan teknologi *Software Defined Network*. *Software Defined Network* adalah sebuah arsitektur jaringan yang memisahkan antara *control plane* dan *data planefast*. Diharapkan pada *Software Defined Network* memiliki *controller* yang menyediakan informasi terkait kondisi jaringan, sehingga proses *failover* nantinya akan semakin cepat. Hal ini ditunjukkan pada hasil pengujian yang mana didapat hasil nilai rata-rata waktu *downtime* sebesar 0,826 detik, nilai rata-rata *failback time* sebesar 0,865 detik, serta nilai rata-rata *packet loss* pada saat proses *failover* terjadi sebesar 5,6%.

Kata kunci: *Client-Server, Failover, High Availability, Software Defined Network (SDN)*

Abstract

Client-Server architecture is an architecture in computer network which involves data exchanges between two devices, in this case client and server. The server role in Client-Server architecture is very important because the server has a task to provide services that required by the client. So, if the server is down, client can't access services that server provides. To solve this problem, we need a high availability system. Many high availability systems which have been built uses master and slave model. In this study, high availability system in Client-Server architecture is built using Software Defined Network technology. Software Defined Network is a network architecture that separates between control plane and data plane. We hope Software Defined Network has a controller which gives information of the network, so the failover process will be faster. We get results of 0.826 second on average downtime, 0.865 second on average failback time, and 5.6% packet loss rate on failover process.

Keywords: *Client-Server, Failover, High Availability, Software Defined Network (SDN)*

1. PENDAHULUAN

Arsitektur *Client-Server* adalah sebuah arsitektur pada jaringan komputer yang melibatkan pertukaran data antar dua perangkat. *Server* merupakan perangkat penyedia layanan dan bertindak sebagai pengelola aplikasi, data, dan keamanan. Sedangkan *client* merupakan perangkat penerima layanan dari *server*, menampilkan, dan menjalankan aplikasi komputer. Dalam arsitektur *Client-Server* dapat menggunakan satu atau beberapa *server* dalam melayani layanan ke banyak *client*. Peranan *server* pada jaringan yang menggunakan arsitektur *Client-Server* sangatlah penting. Karena, *server* bertugas untuk menyediakan layanan yang dibutuhkan oleh *client*. Maka, ketika terjadi gangguan pada *server* hingga menyebabkan *server* tersebut *down*, akibatnya akan menyebabkan keseluruhan layanan yang dibutuhkan oleh *client* tidak tersedia. Beberapa contoh aplikasi yang menggunakan Arsitektur *Client-Server* diantaranya adalah web, *File Transfer Protocol* (FTP), *telnet*, dan e-mail (Kurose & Ross, 2013).

Salah satu langkah yang dapat dilakukan untuk mengatasi permasalahan tidak tersedianya

layanan yang diberikan oleh *server* adalah dengan mengimplementasikan *High availability Clusters* (HAC). Tujuan dari HAC adalah untuk menjaga ketersediaan layanan yang diberikan oleh *server* utama (*master server*) kepada *client*. Hal tersebut dapat dilakukan dengan memasang perangkat lunak cluster pada *master server* dan *backup server* (*slave server*) (Vugt, 2014). *Master server* bertugas untuk menyediakan layanan bagi *client*, dimana tugas tersebut akan dijalankan *slave server* apabila *master server* mengalami *down*. Perpindahan dari *master server* ke *slave server* disebut dengan *failover*. Waktu *failover* dapat bervariasi sesuai dengan konfigurasi *server* yang ada pada jaringan (Jayaswal, 2006).

Pada penelitian sebelumnya oleh Parwitayasa (2012) peneliti membahas tentang *Heartbeat* dan *Distributed Replicated Block Device* (DRBD) untuk sistem *High Availability* pada *server*. *Heartbeat* merupakan sebuah sistem berbasis *High Availability* yang menerapkan serial UDP dan *Point to Point Protocol* (PPP). Sedangkan DRBD merupakan perangkat lunak yang berfungsi untuk mereplikasi database dan aplikasi (LinBit, 2016). Hasil dari penelitian tersebut didapat sebuah sistem *High availability server* dengan *downtime* berkisar antara 7 hingga 8 detik untuk perpindahan dari *master* ke *slave*.

Pada penelitian kali ini, peneliti menggunakan Arsitektur Software Defined Network untuk melakukan penelitian tentang "Implementasi *Server Failover* Pada *Software Defined Network*". *Software Defined Network (SDN)* adalah sebuah arsitektur jaringan komputer yang memisahkan antara *control plane* dan *data plane*. Untuk mengatur *control plane* dengan *data plane* pada arsitektur jaringan SDN menggunakan sebuah unit pengontrol yang disebut *controller*. *Control plane* merupakan komponen jaringan yang berfungsi sebagai pengontrol sedangkan *data plane* berfungsi meneruskan paket data yang masuk suatu *port* menuju *port* dengan komunikasi pada *control plane* (Laksana, 2016). Cara berkomunikasi antara perangkat dengan *controller* menggunakan sebuah protokol yang disebut dengan *OpenFlow*. *OpenFlow* merupakan seperangkat protokol dan *Application Programming Interface* (API) yang digunakan pada arsitektur jaringan SDN (Gray, 2013). Kelebihan menggunakan arsitektur SDN adalah *Programmability* dan *Visibility*. *Programmability* adalah kemampuan untuk mengubah perilaku jaringan serta untuk melakukan perubahan secara otomatis. *Visibility* adalah kemampuan untuk dapat memonitor jaringan, baik dari sisi sumber daya, dan konektivitas.

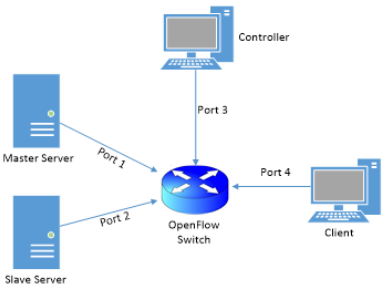
Pada penelitian ini menggunakan sebuah *controller* berbasis *framework Ryu* dan menggunakan pemrograman *Python* untuk mengenali setiap *server* dan *client* yang ada pada jaringan dengan menerapkan mekanisme *failover* untuk mengatasi kegagalan yang terjadi pada *server*. Tujuan yang ingin dicapai dengan menggunakan arsitektur SDN ini adalah untuk mengurangi waktu *downtime* dan waktu *failback server* dengan memanfaatkan *controller* berbasis *framework Ryu* tersebut.

2. PERANCANGAN DAN IMPLEMENTASI

Pada perancangan kali ini peneliti akan menjabarkan semua perancangan, dimana pada perancangan tersebut terdiri dari perancangan topologi, perancangan *server*, perancangan *switch*, perancangan *controller*, dan perancangan *client*.

2.1 Perancangan Topologi

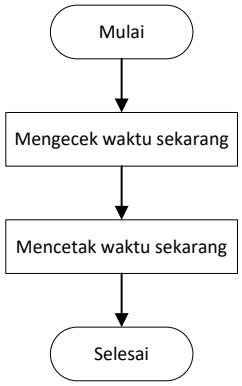
Pada Gambar 1 menjelaskan topologi jaringan yang digunakan pada penelitian ini. Dimana pada penelitian ini akan mengimplementasikan sebuah arsitektur SDN dengan menggunakan dua buah *server* yang telah dikonfigurasi dan siap untuk melayani *client*, sebuah *controller* dan sebuah switch yang nantinya akan mengarahkan *request* dari *client* ke *server*. *Switch* tersebut nantinya akan dikonfigurasi untuk meneruskan paket melalui 4 buah *port* yang berbeda. *Interface eth0* pada port pertama terhubung dengan *master server*, *interface eth1* pada port kedua terhubung dengan *slave server*, *interface eth2* pada port ketiga terhubung dengan *controller*, dan *interface eth3* pada port keempat terhubung dengan *client*.



***Gambar 1** Perancangan Topologi*

2.2 Perancangan *Server*

Penggunaan *server* pada penelitian ini berfungsi sebagai web *server* yang dibangun menggunakan aplikasi Apache. Setelah instalasi *server* dilakukan, maka akan dilakukan proses konfigurasi pengalamatan alamat IP dan MAC Address. Lalu ditambahkan sebuah program dengan menggunakan bahasa pemrograman *Python* Program tersebut berfungsi untuk mencetak paket yang telah sampai *server* yang dikirimkan terus-menerus oleh *client*. Berikut Gambar 2 menunjukkan program *flowchart* untuk mrnampilkan waktu pada *server*.



Gambar 1 *Flowchart* Menampilkan Waktu Paket

Berikut Tabel 1 akan menunjukkan spesifikasi *server*.

Table 1 Spesifikasi *Server*

Master Server	Slave Server
Sistem Operasi : Ubuntu *Server* 14.04 LTS x64 Swap size : 8GB **IP Address:** 10.0.0.1 **MAC Address:** 00:01:6C:55:71:77	**Sistem Operasi :** Ubuntu *Server* 14.04 LTS x64 Swap size : 8GB **IP Address:** 10.0.0.2 **MAC Address:** 00:01:6C:55:71:3B

2.3 Perancangan Switch

Sebuah arsitektur SDN minimal dibutuhkan sebuah *switch* yang mendukung penggunaan protokol *OpenFlow*. *Switch* diimplementasikan dengan menggunakan aplikasi *Open vSwitch*. *Open vSwitch* merupakan sebuah aplikasi yang digunakan untuk membangun sebuah virtual *switch* pada sistem operasi *Linux*. *Open vSwitch* juga dapat diimplementasikan pada komputer desktop pada umumnya. Pada penelitian ini *switch* dibangun pada sebuah komputer dengan sistem operasi *Linux Ubuntu* dan menggunakan aplikasi *Open vSwitch 2.8.2*. Pada komputer yang digunakan terdapat 4 buah *network interface* yang digunakan sebagai *port* dari *Open vSwitch*. Berikut Tabel 2 menunjukkan pemetaan *network interface* dan *port Open vSwitch*.

Table 2 Pemetaan *Network Interface* Dan *Port Open vSwitch*

Interface	Port	Tujuan
eth0	Port 1	*Master Server*
eth1	Port 2	*Slave Server*
rename4	Port 3	*Controller*
rename5	Port 4	*Client*

Berikut Tabel 3 akan menunjukkan spesifikasi *server*.

Table 3 Spesifikasi Switch

Switch		
Hardware	**Sistem Operasi**	**IP Address**
Lenovo 3000 H Series Intel Dual Core E2220 / 2.4 Ghz 2GB RAM 320 GB SATA 7200 RPM	Ubuntu *Server* 14.04 LTS x64 Swap size : 8GB	10.0.0.6

2.4 Perancangan Controller

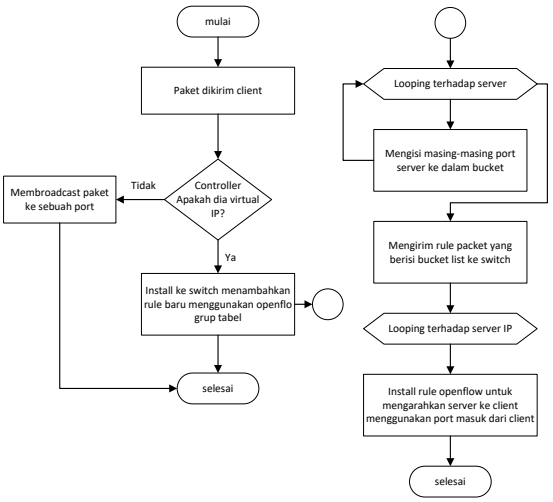
Pada penelitian ini *server failover* akan dibuat pada aplikasi *controller Ryu*. *Controller* akan bekerja untuk memandu *switch* untuk meneruskan paket dan disaat yang sama juga menjadi *server failover* yang berfungsi untuk mengecek ketersediaan *server* aktif dan apabila terjadi kesalahan dari salah satu *server* maka jalur data akan dialihkan ke *server* yang lainnya. *Ryu* yang digunakan adalah *Ryu 4.12*. Berikut Tabel 4 akan menunjukkan spesifikasi *controller*.

Table 4 Spesifikasi *Controller*

Controller		
Perangkat Keras	**Sistem Operasi**	**Alamat IP**
ASUS A455LN Intel Core i3-4210U / 1.80 Ghz 8192 MB 500 GB HDD	Ubuntu *Server* 14.04 64-bit Swap : 4096 MB	10.0.0.5

2.4.1 Perancangan *Failover*

Failover berdasarkan status *server* yang bekerja diatas aplikasi *Ryu*. Modul untuk melakukan pengecekan terhadap *server* dijalankan oleh sebuah program *failover*. Berikut adalah alur *flowchart* program *failover*.

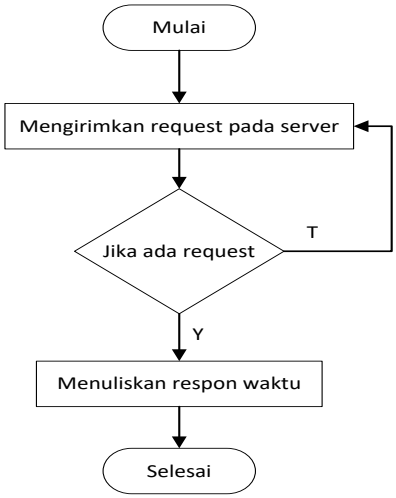


Gambar 2 *Flowchart* Program *Failover*

Pada Gambar 3 dijelaskan bahwa program berjalan mulai dengan paket dikirim oleh *client* ke *switch* kemudian *controller* mengecek tujuan paket dikirim ke virtual IP atau tidak. Apabila bukan virtual IP maka akan menjalankan perintah untuk membroadcast paket ke sebuah *port* dan program selesai bekerja, jika tujuan paket sampai virtual IP maka akan menjalakan perintah install ke *switch* dan menambahkan rule baru menggunakan *controller* grup tabel. Dalam perintah tersebut terdapat perintah baru lagi yaitu melakukan looping terhadap *server* dengan mengisi masing-masing *port* ke dalam *bucket*. Setelah selesai melakukan pengisian ke dalam *bucket* lalu mengirimkan rule packet yang berisi *bucket list* ke *switch*. Kemudian melakukan looping terhadap *server* IP atau Virtual IP lalu menjalankan perintah install rule *controller* untuk mengarahkan dari *server* ke *client* menggunakan *port* masuk dari *client*, dengan begitu program selesai bekerja.

2.5 Perancangan *Client*

Pada Gambar 4 dijelaskan bahwa alur program *request* paket yang dijalan pada *client*. Dimulai *client* mengirimkan *request* pada *server* kemudian muncul perulangan jika *request* dari *client* tidak diterima oleh *server* maka program akan terus menurus melakukan perulangan meminta *request* kepada *server* sampai ada balasan dari *server*. Apabila *request* dari *client* ada telah diterima oleh *server* maka program akan menuliskan respon waktu data sampai ke *server*. Setelah itu program selesai berjalan.



Gambar 3 *Flowchart* Request Paket

2.6 Implementasi

Pada implementasi menerapkan semua rancangan. Proses instalasi yang dilakukan dalam pembuatan arsitektur SDN. Pada arsitektur SDN yang dibuat terdiri dari *server* yang dipasang layanan *web service* sebagai penyedia layanan kepada *client*, *switch* yang dipasang aplikasi *Open vSwitch* untuk menghubungkan antara *client* dengan *server*, *controller* yang dipasang *framework Ryu* untuk menjalankan sebuah program yang mengatasi *failover*, dan *client* yang dipasang aplikasi *Iperf* untuk melakukan pengujian *packet loss* dari *server* ke *client*.

3. PENGUJIAN

Setelah melalui tahap perancangan dan implementasi, maka tahap selanjutnya adalah menguji sistem yang telah dibangun. Proses pengujian meliputi pengujian fungsionalitas program *failover* dan pengujian kinerja *High availability server* yang terdiri dari parameter *downtime*, *failback time*, dan *packet loss*.

3.1 Pengujian Fungsionalitas Program *Failover*

Pengujian fungsionalitas program *failover* dilakukan dengan melakukan proses ping test untuk mengetahui jaringan yang ada apakah terkoneksi dengan bai atau tidak. Proses tersebut dilakukan dengan mengirim pesan *request* dari *client* menuju *server* dengan melakukan ping ke alamat virtual IP terlebih dahulu. Alamat virtual IP yang digunakan adalah 10.0.0.100. Setelah itu, kemudian *client* membuka *browser* lalu mengakses alamat virtual IP tersebut. Hasilnya virtual IP akan menampilkan halaman web dari *master server*. Karena pada program *failover* tersebut telah diatur *master server* bertugas menjadi *server* utama. Selama *master server* aktif, semua *client* yang mengakses virtual IP akan diarahkan menuju *master server*. Kemudian untuk dilakukan pemutusan pada jalur menuju *master server* dengan cara mematikan *port* pada *switch* yang terhubung pada *master server*. Pada saat *master server* mengalami jalur data *down* atau mati maka *controller* akan mengalihkan jalur komunikasi dari *master server* ke *slave server*. Kemudian ketika jalur data *master server* terhubung kembali, maka ketika pengguna mengakses virtual IP akan diarahkan kembali ke *master server*.

Hasil pengujian fungsionalitas didapatkan beberapa hasil yaitu hasil perpindahan *website* dari *master server* ke *slave server* begitupun sebalikan pada proses *failback*, Perubahan tersebut diatur oleh *switch* yang didalamnya terdapat protokol *OpenFlow* dimana terdapat mekanisme *fast-failover*. Keberhasilan dari pengujian diperlihatkan dari Gambar 5 dibawah ini yang menunjukkan hasil perpindahan dari *master server* ke *slave server* tanpa merubah alamat IP.

Gambar 4 Perintah *Down Master Server* dan *Client* Akses Web

3.2 Pengujian Kinerja *High Availability Server*

Pada pengujian kinerja *High availability server* yang dibangun terdapat tiga parameter yang diuji meliputi pengujian *downtime*, *failback time*, dan *packet loss*.

3.2.1 Pengujian *Downtime*

Pengujian *Downtime* dilakukan untuk mengetahui waktu perpindahan dari *master server* ke *slave server* dengan cara *client* menjalankan sebuah program yang nantinya program tersebut akan mengirimkan paket *request* ke *server* secara terus menerus yang isinya meminta waktu *server* pada saat itu. Ketika *master server* aktif, maka *master server* tersebut yang akan melayani *request* dari *client* dan pada tampilan log di *server* akan menampilkan data berupa waktu *server* saat itu sekaligus mengirimkan data tersebut ke *client*. Ketika *master server down*, maka proses tersebut akan dilakukan oleh *slave server*. Log waktu terakhir yang muncul di *master server* dan log waktu pertama yang muncul di *slave server* akan digunakan untuk perhitungan waktu *Downtime*. Caranya dengan mengurangi log waktu pertama di *slave server* dengan log waktu terakhir di *master server*. Pada Tabel 5 dibawah ini merupakan hasil pengujian *Downtime* yang didapat setelah dilakukan 10 kali percobaan.

Table 5 Rata-Rata Pengujian *Downtime*

Pengujian Downtime	
Pengujian	Waktu (s)
1	1.01
2	1.06
3	0.77
4	0.7
5	1
6	0.5
7	0.83
8	0.91
9	0.57
10	0.91
Rata-rata	0.826

Dari hasil pengujian yang telah dilakukan pada Tabel 5 memperlihatkan hasil dari pengujian *downtime* yang telah dilakukan, dimana didapat hasil waktu *downtime* tertinggi pada percobaan ke-2 selama 1,06 detik dan terendah pada percobaan ke-6 selama 0,5 detik. Sedangkan rata-rata waktu *downtime* yang didapatkan selama 0,826 detik.

3.2.2 Pengujian *Failback Time*

Pengujian *failback time* merupakan kebalikan dari *downtime*, yang berarti *failback time* merupakan waktu perpindahan dari *slave server* ke *master server*. Perhitungan *failback time* dilakukan dengan cara mengurangi log waktu pertama di *master server* dengan log

waktu terakhir di *slave server*. Pada Gambar 7 dan Tabel 6 dibawah ini merupakan hasil pengujian *downtime* yang didapat setelah dilakukan 10 kali percobaan.

Table 6 Rata-Rata Pengujian *Failback Time*

Pengujian *Failback Time*	
Pengujian	Waktu (s)
1	1.12
2	1.05
3	0.56
4	0.8
5	1.1
6	0.55
7	0.73
8	0.85
9	0.93
10	0.96
Rata-rata	0.865

Dari hasil pengujian yang telah dilakukan pada Tabel 6 memperlihatkan hasil dari pengujian *failback time* yang telah dilakukan, dimana didapat hasil *failback time* tertinggi pada percobaan ke-1 selama 1,12 detik dan terendah pada percobaan ke-6 selama 0,55 detik. Sedangkan rata-rata waktu *Downtime* yang didapatkan selama 0.865 detik.

3.2.3 Pengujian *Packet Loss*

Pengujian *packet loss* bertujuan untuk mengetahui berapa banyak data yang hilang pada saat sistem melakukan pengiriman data. Pengujian ini dilakukan sebelum dan pada saat program *failover* berkerja. Aplikasi yang digunakan untuk menguji *packet loss* adalah *Iperf* dengan dengan menjalankan perintah *iperf –s –u* di sisi *server*. Perintah –s menandakan bahwa perintah dijalankan di sisi *server* dan –u menandakan pengiriman data menggunakan protokol UDP. Perintah tersebut akan dijalankan di *master server* dan *slave server*. Sedangkan di sisi *client* digunakan perintah –c. Pada Tabel 7 dan Tabel 8 dibawah ini merupakan hasil pengujian *packet loss* yang didapat setelah dilakukan percobaan sebanyak 10 kali dimana satu kali percobaan dilakukan selama 20 detik.

Table 7 Rata-Rata Pengujian *Packet Loss* Sebelum *Failover* Bekerja

Pengujian Packet Loss		
Pengujian	Waktu (s)	Packet Loss (%)
1	20	0
2		0
3		0
4		0
5		0
6		0
7		0
8		0
9		0
10		0
Rata-rata		0

Table 8 Rata-Rata Pengujian Packet Loss pada saat Program Failover Bekerja

Pengujian Packet Loss		
Pengujian	Waktu (s)	Packet Loss (%)
1	20	6.3
2		5
3		6.3
4		4.4
5		6.6
6		6.3
7		6.8
8		4.1
9		4
10		6.2
Rata-rata		5.6

Dari hasil pengujian yang telah dilakukan pada Tabel 7 dan Tabel 8 memperlihatkan hasil dari pengujian *packet loss* yang telah dilakukan, dimana didapat hasil rata-rata *packet loss* sebelum *failover* bekerja dari 10 kali percobaan dalam waktu 20 detik didapatkan hasil 0%. Sedangkan dapat dilihat pada Tabel 8 menunjukkan hasil rata-rata *packet loss* pada saat *failover* bekerja dari 10 kali percobaan dalam waktu 20 detik didapatkan hasil 5,6%.

4. KESIMPULAN

Dalam implementasinya pada penelitian ini menggunakan dua *server*, satu *controller*, satu *switch*, dan satu *client*. Peneliti membuat program berbasis *Python* untuk mengatasi *failover* pada *server*. Program tersebut dijalankan pada *switch* melalui sebuah *controller*. Didalam *switch* terdapat sebuah protokol yang disebut *OpenFlow*. Setelah dijalankan, virtual IP menginisialisasi IP *server* sehingga *client* tetap dapat mengakses *server* meskipun terjadi masalah pada *server* utama.

Slave server berperan megambil alih layanan dari *master server* apabila *master server* mengalami masalah sehingga *client* tetap bisa mendapatkan layanan dari *server*. Setelah *master server* aktif kembali, layanan yang diambil alih *slave server* akan dikembalikan ke *master server*. Ketika *client* mengakses sebuah *server*, virtual IP yang menghandel *request client* lalu diteruskan menuju *server*. Setelah *request client* diterima oleh *server*, lalu *server* membalas *request* dengan mengirimkan paket yang diterima oleh virtual IP yang lalu diteruskan menuju *client*. Sehingga *controller* dapat mengatur pertukaran data yang terjadi antara *client* dengan *server*.

Dalam penelitian ini didapatkan hasil nilai rata-rata *downtime* sebesar 0,826 detik dari 10 kali percobaan yang dilakukan peneliti. Hasil nilai rata-rata *failback time* sebesar 0,865 detik dari 10 kali percobaan. Pada percobaan *packet loss* dilakukan pengujian sebelum dan pada saat *failover* terjadi, yang hasil nilai *packet loss* sebelum terjadi *failover* rata-rata yang didapat adalah 0%. Kemudian hasil nilai *packet loss* pada saat *failover* terjadi rata-rata yang didapat adalah 5,6%.

5. DAFTAR PUSTAKA

Jayaswal, K., 2006. *Administering Data Centers Server, Storage, and Voice Over IP*. Indianapolis: Wiley Publishing, Inc..

Kurose, J. F. & Ross, K. W., 2013. *Computer Networking*. Sixth ed. New Jersey: Addison Wesley.

Nadean, K. G. &. T. D., 2013. *SDN: Software Defined Network*. First ed. United States of America: O'Reilly Media.

Vugt, S. v., 2014. *Pro Linux High Availability Clustering*. New York: Heinz Weinheimer.

Gibbs, M., 2005. *High availability and Heartbeat | Netwrok World*. [Online] Tersedia di:<http://www.networkworld.com/article/2321687/software/high-availabilityandheartbeat.html>[Diakses 17 Maret 2017].

Gray, T. D. N. &. K., 2013. An Authoritative Review of Network Programmability Technologies. In: M. L. a. M. Blanchette, ed. *Software Defined Nework* . United States of America: O'Reilly Media, pp. 47-49.

Laksana, E. K. D., 2016. Analisis *Controller* load balancing Web *Server* Dengan Algoritma Least Connection Pada Software Defined Network. *Jaringan Komputer*, 12 Agustus, p. 1.

LinBit, 2016. *DRBD HA*. [Online] Tersedia di: <https://www.linbit.com/en/ products-and-services/drbd/> [Diakses 17 Maret 2017].

Parwitayasa, R. W., 2012. *Penerapan Sistem High Availability Pada Server Teknik Informatika Universitas Brawijaya Menggunakan Aplikasi Heartbeat dan Distributed Replicate Block Device (DRBD)*. 1 ed. Malang: Fakultas Ilmu Komputer.