IMPLEMENTASI SERVER FAILOVER PADA SOFTWARE DEFINED NETWORK

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Dea Asmara Gita
NIM: 135150301111034

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

PENGESAHAN

IMPLEMENTASI SERVER *FAILOVER* PADA *SOFTWARE DEFINED NETWORK*

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Dea Asmara Gita
NIM: 135150301111034

Skripsi ini telah diuji dan dinyatakan lulus pada
30 Juli 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Sabriansyah Rizqika Akbar, S.T., M.Eng.	Widhi Yahya, S.Kom., M.Sc.
NIP: 19820809 201212 1 004	NIK: 201607 891121 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D
NIP: 19710518 200312 1 001

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, 20 Juli 2018

Dea Asmara Gita

NIM: 135150301111034

KATA PENGANTAR

Syukur Alhamdulillah penulis ucapkan atas kehadirat Allah Subhanahu wa Ta'ala, karena berkat rahmat dan anugerah-Nya penulis dapat menyelesaikan skripsi yang berjudul "Implementasi *Server Failover* Pada Software Defined Network". Shalawat serta salam semoga selalu tercurahkan pada baginda Rasulullah Muhammad SAW. Penulisan skripsi ini akan sulit terwujud apabila tidak ada rahmat dari Allah subhanahu wa Ta'ala serta bantuan dan dukungan dari berbagai pihak. Oleh karena itu, penulis ingin mengucapkan rasa terima kasih kepada :

1. Yang terhormat Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya
2. Yang terhormat Bapak Tri Astoto Kurniawan, S.T, M.T, Ph,D selaku Ketua Jurusan Teknik Informatika yang telah memeberikan kesempatan untuk mengikuti ujian skripsi
3. Yang terhormat Bapak Sabriansyah Rizqika Akbar, S.T, M.Eng. selaku Kepala Program Studi Teknik Komputer dan pembimbing skripsi 1 yang selalu dengan senang hati memberikan bimbingan kepada penulis dalam melakukan penelitian skripsi
4. Yang terhormat Bapak Widhi Yahya, S.Kom., M.Sc selaku pembimbing skripsi 2 yang selalu dengan sabar memberikan bimbingan kepada penulis dalam melakukan keseluruhan penelitian
5. Ibu Ees Sukaesih yang selalu memberikan dukungan nasehat, semangat, motivasi, doa, dan memberikan canda dan tawanya dalam suka duka menjalani kuliah maupun pengerjaan skripsi
6. Kepada Febbi Ulul Fadila, S.Pd atas kesabaran dan dukungan luar biasa kepada penulis untuk menyelesaikan penelitian ini
7. Kepada Keluarga Besar Penulis, Keluarga Kost Joyosuko 61A, Andy Jaya H, S.Kom, dan Bella Aulia Rahmataufany, S.Kom, serta teman-teman seperjuangan lainnya yang telah memberikan dukungan moril dan semangat kepada penulis

Semoga skripsi ini dapat bermanfaat bagi penulis dan orang lain yang membaca skripsi ini

Malang, 20 Juli 2018

Dea Asmara Gita

deaa.asmara@gmail.com

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 plane*. 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.

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

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Arsitektur *Client-Server* adalah sebuah arsitektur pada jaringan komputer yang melibatkan pertukaran data antar dua perangkat. *Server* merupakan perangkat yang menyediakan layanan dan bertindak sebagai pengelola aplikasi, data, dan keamanan. Sedangkan C*lient* adalah perangkat yang menerima layanan, 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 s*lave 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 yang terkait *failover* yaitu penelitian yang berjudul "*Penerapan Sistem High Availability Pada Server Teknik Informatika Universitas Brawijaya Menggunakan Aplikasi Heartbeat dan Distributed Replicate Block Device (DRBD)"* (Parwitayasa, 2012). Pada penelitian tersebut membahas tentang *heartbeat* dan *distributed replicated block device* (DRBD). *Heartbeat* merupakan sebuah basis dari *HA* yang mengimplementasikan serial UDP, dan *Point to Point Protocol* (PPP) *heartbeat* bersamaan dengan pengambilan alamat *IP* termasuk *resources model* dan *resources group* (Gibbs, 2005). Sedangkan *DRBD* adalah perangkat lunak yang berguna untuk mereplika *database* dan aplikasi (LinBit, 2016). Hasil penelitian sebelumnya menggunakan *heartbeat* sebagai sistem *high availability server* dengan *downtime* berkisar 7 sampai 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.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana rancangan sistem *high availability server* pada *Software Defined Network*?
2. Bagaimana kinerja sistem *high availability server* ditinjau melalui parameter *downtime*, *failback time*, dan *packet loss* menggunakan *Software Defined Network*?

1.3 Tujuan

Dari rumusan masalah penelitian ini bertujuan untuk:

1. Membangun sebuah sistem pada *SDN* yang memiliki *high availability server* dengan implementasi pada sebuah perangkat asli.

2. Melakukan pengujian kinerja sistem dengan melakukan percobaan terhadap fungsionalitas, *downtime, failback time*, dan *packet loss*.

1.4 Manfaat

Dalam penelitian ini diharapkan bermanfaat dalam hal sebagai berikut ini:

1. Dapat melakukan implementasi sistem *high availability server* sehingga dapat diketahui hasil ketersediaan dari *server*.
2. Dapat mengetahui kinerja hasil dari *downtime*, *failback time, dan packet loss* pada *SDN*.
3. Dapat mengetahui performa suatu sistem *high availability server* dari hasil analisis yang telah dikerjakan.

1.5 Batasan Masalah

Batasan masalah yang dapat dipetik berdasarkan rumusan masalah adalah sebagai berikut:

1. Menggunakan *Ryu* sebagai *framework* dan bahasa pemrograman *Python* pada *Software Defined Network*.
2. Penelitian dilakukan dengan menggunakan dua buah *web server*, satu buah *switch*, satu buah *controller*, dan *client*.
3. Sistem *failover* bekerja ketika *server* mengalami *down* atau mati, jika aplikasi dalam sebuah *server* mengalami *down* atau kegagalan maka sistem *failover* tidak bekerja.

1.6 Sistematika Pembahasan

Langkah-langkah atau tahapan-tahapan yang akan dilakukan dalam menyelesaikan skripsi ini adalah sebagai berikut:

BAB I PENDAHULUAN

Pada bab ini menguraikan latar belakang masalah dari penelitian, rumusan masalah bagaimana penerapan program *failover*. Memiliki tujuan mengimplementasikan *server failover* pada arsitektur *Software Defined Network.* Menjelaskan manfaat dari sistem untuk mengembangkan mekanisme *fast-failover* yang dapat diimplementasikan pada jaringan dengan arsitektur *Software Defined Network* (SDN). Menjelaskan sistematika penulisan yang digunakan dalam penyusunan tugas akhir.

BAB II LANDASAN KEPUSTAKAAN

Pada bab ini menguraikan dasar teori dan teori penunjang yang berkaitan dengan penelitian meliputi *high availability server* pada arsitektur *Software Defined Network* (SDN).

BAB III METODOLOGI PENELITIAN

Pada bab ini menjelaskan metode yang digunakan terdiri dari studi literatur, analisa kebutuhan, implementasi, pengujian dan hasil, serta pengambilan kesimpulan dan saran.

BAB IV REKAYASA KEBUTUHAN

Pada bab ini membahas tentang kebutuhan fungsional dan non fungsional dari implementasi *server failover* pada jaringan dengan arsitektur *Software Defined Network* (SDN).

BAB V PERANCANGAN DAN IMPLEMENTASI

Pada bab ini membahas tentang perancangan sistem dan implementasi *server failover* yang diimplementasikan pada jaringan dengan arsitektur *Software Defined Network* (SDN).

BAB VI PENGUJIAN DAN HASIL

Pada bab ini membahas tentang pengujian *server failover* yang diimplementasikan pada jaringan dengan arsitektur *Software Defined Network* (SDN) serta analisa dari hasil pengujian tersebut.

BAB VII PENUTUP

Pada bab ini berisi kesimpulan yang menjawab rumusan masalah dari penelitian ini serta saran untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Kajian pustaka dalam penelitian ini membahas beberapa perbandingan penelitian sistem pendukung keputusan yang sudah dilakukan sebelumnya, pembahasan tentang penelitian sebelumnya digunakan penulis sebagai acuan untuk mendukung penulisan ini.

Tabel 2.1 Kajian Pustaka

No.	Nama Penulis, Tahun, dan Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1.	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.	Sistem *high availability server*	Menggunakan aplikasi *heartbeat* dan *Distributed Replicate Block Device*	Pengembangan sistem high availability *server* menggunakan *software define network*
2.	Hernawan Sulistyanto, S. M., 2015. *IMPLEMENTASI HIGH AVAILABILITY SERVER DENGAN TEKNIK FAILOVER VIRTUAL COMPUTER CLUSTER.* Surakarta: Universitas Muhammadiyah Surakarta.	Sistem *high availability server*	Menggunakan tekni*k failover virtual computer cluster*	Pengembangan sistem *high availability server* menggunakan *software define network*

Pada penelitian yang telah dilakukan sebelumnya untuk mengembangan sistem *high availability server* pada jaringan tradisional beberapa peneliti telah mengusulkan beberapa penelitian. Penelitian-penelitian tersebut kemudian menjadi dasar dalam penyusunan penelitian ini. Parwitayasa pada penelitiannya yang berjudul "Penerapan Sistem *High Availability* Pada *Server* Teknik Informatika Universitas Brawijaya Menggunakan Aplikasi *Heartbeat* dan *Distributed Replicate Block Device* (DRBD)" mengusulkan untuk melakukan pembuatan sistem HA *server*

dengan aplikasi heartbeat dan DRBD. Aplikasi tersebut melakukan kinerja dalam mengatasi *failover* dan *failback* dengan cara menggunakan port UDP 694 sebagai jalur komunikasi antar *server*, dan pengecekan pada setiap *server* dilakukan secara broadcast pada satu jaringan *IP adddress* yang telah dialokasikan. Hasil aplikasi tersebut memerlukan waktu downtime berkisar 7 sampai 8 detik (Parwitayasa, 2012). Implementasi penelitian tersebut dilakukan dengan installasi aplikasi pada setiap server. Peneliti tidak menjelaskan jumlah *packet loss* yang terjadi saat *failover.*

Pada penelitian Hernawan Sulistyanto yang berjudul "*Implementasi High Availability Server Dengan Teknik Failover Virtual Computer Cluster*". Dalam penelitiannya dijelaskan bahwa aplikasi yang digunakan adalah *heartbeat* dan DRBD juga tetapi menggunakan teknik *failover virtual computer cluster*. Peneliti melakukan implementasi menggunakan aplikasi virtualisasi yaitu *VMware Workstation 11*. Hasilnya aplikasi tersebut dapat memerlukan waktu *downtime* selama 4 detik (Sulistyanto, 2015). Namun implementasi dari penelitian tersebut masih kurang karena implementasi tidak dilakukan secara nyata pada hardware langsung tetapi menggunakan virtualisasi. Hal ini menciptakan kemungkinan untuk hasil tidak mendapatkan secara maksimal dalam pengujian yang dilakukan oleh peneliti.

2.2 Dasar Teori

Landasan teori yang digunakan dalam penelitian ini berasal dari jurnal-jurnal penelitian sebelumnya serta literatur-literatur yang menunjang. Teori yang digunakan sebagai referensi tidak diambil secara keseluruhan dari jurnal. Hanya teori-teori yang dijadikan sebagai dasar penelitian yang penulis masukkan kedalam laporan ini. Landasan-landasan teori yang diambil yaitu berkaitan dengan *Software Defined Network* (SDN) meliputi Mekanisme, Protokol serta sistem pendukung, sistem *high availability server* pada SDN.

2.2.1 Software Defined Network

Software Defined Network (SDN) adalah sebuah paradigma baru dalam pengontrolan dan manajemen jaringan komputer. Konsep dasar dari SDN adalah pemisahan antara *control plane* yang bertugas untuk menentukan bagaimana sebuah paket akan diteruskan dengan data *plane* yang bertugas meneruskan paket (Anees Al-Najjar, 2016). Menurut Nadeu & Gray dalam bukunya "*SDN: Software Defined Network*", SDN adalah pendekatan arsitektural yang memberikan optimasi serta menyederhanakan operasi di dalam jaringan dengan lebih mendekatkan interaksi antara aplikasi, *network service*, serta *network device* pada jaringan asli maupun *virtual*. Hal tersebut dapat dicapai dengan mengimplementasikan sebuah *logical server* yang tersentralisasi (dalam hal ini bisa disebut SDN *controller*) yang mengatur, memediasi, dan memfasilitasi aplikasi untuk berkomunikasi dengan perangkat jaringan, dan perangkat jaringan yang ingin mengirimkan informasi menuju aplikasi (Nadean, 2013). Komunikasi antar aplikasi dilakukan dengan menggunakan protokol standar yang disebut

Controller (OF) (Nick Mckeown, 2008). Arsitektur SDN tergolong baru sehingga memberikan ruang yang luas untuk penelitian dan pengembangan. Secara umum bagian dari SDN dapat dikategorikan menjadi 3 bagian, yaitu *control plane*, *data plane*, dan *application plane*, serta protokol yang mengatur komunikasi antara *controller* dengan *data plane* yaitu *OpenFlow*. Berikut adalah penjelasan singkat mengenai beberapa bagian tersebut.

2.2.1.1 Control Plane

Control plane adalah bagian dari SDN yang membentuk *forwarding table* berdasarkan data set yang dibentuk dan disimpan pada *control plane*. *Forwarding table* tersebut kemudian akan digunakan oleh *data plane* untuk meneruskan *traffic* antara *ingress* dan *engress port* pada suatu *device*. *Dataset* yang digunakan untuk menyimpan topologi jaringan disebut *Routing Information Base* (RIB). Setelah RIB terbentuk dan stabil, RIB tersebut kemudian akan digunakan untuk memprogram *Forwarding Information Base* (FIB) yang digunakan oleh *data plane* untuk meneruskan paket (Nadean, 2013). Dalam implementasinya *server* dapat bekerja dengan tiga mode, *reactive mode*, *proactive mode*, dan *hybrid mode* (Fernandez, 2013). Mode tersebut bekerja sebagai berikut :

- *Reactive mode* : Pada *reactive mode*, ketika sebuah paket tiba pada sebuah *network device* (*switch*, dll.), *switch* akan melihat *flow rule* di dalam *flow table* miliknya. Jika tidak terdapat aturan yang sesuai, paket akan diteruskan menuju *controller*. *Controller* kemudian akan menentukan kemana paket akan diteruskan. Setelah selesai, *controller* akan mengubah *flow table* pada *switch* berdasarkan paket sebelumnya.
- *Proactive mode* : Pada *proactive mode*, *flow entries* pada *flow table* telah diatur sebelumnya. Jadi ketika ada paket yang datang, *switch* telah mengetahui kemana paket akan diteruskan. *Controller* tidak ikut campur dalam menentukan *flow rule*.
- *Hybrid mode* : Pada *hybrid mode*, *controller* mengambil keuntungan dari *reactive* dan *proactive mode*. *Controller* dapat mengatur *flow rule* secara *proactive* dan *reactive*.

Control layer terdiri dari satu atau lebih SDN *controller* yang mengimplementasi sebuah *Network Operating System* (NOS) yang berfungsi untuk mengatur data plane yang terhubung serta menyediakan jalur komunikasi antar device yang terhubung di dalam jaringan. NOS juga berfungsi menyediakan informasi bagi aplikasi yang berada di layer yang lebih tinggi (Anees Al-Najjar, 2016). Pada penelitian ini NOS yang digunakan adalah *Ryu*. *Ryu* dipilih karena merupakan NOS yang banyak diimplementasikan pada *OpenStack cloud operating system* yang banyak digunakan pada *cluster server* dan *data center* (Shie-Yuan Wang, 2015). Terlebih lagi aplikasi yang berbasis *Ryu* ditulis dalam bahasa pemrograman *Python* sehingga mudah diterapkan pada Sistem Operasi berbasis *UNIX*.

2.2.1.2 Data Plane

Data plane adalah bagian dari SDN yang berfungsi menangani *datagram* (pada kabel, *fiber*, atau *wireless*) melalui serangkaian operasi yang dilakukan pada *link-level*. *Datagram* tersebut kemudian akan diteruskan berdasarkan tabel FIB yang yang telah diprogram oleh SDN *controller* (Nadean, 2013). Data plane merupakan lapisan terendah dari SDN yang berisi *network device* seperti *router, phisical/virtual switch, access point* dll. *Data plane* dapat berkomunikasi dengan SDN *controller* menggunakan protokol *OpenFlow* (Durresi, 2017).

2.2.1.3 *OpenFlow* Protokol

OpenFlow adalah *interface* komunikasi standar yang menjelaskan mekanisme komunikasi antara *controller layer* dan *forwarding layer* pada arsitektur SDN. *OpenFlow* menyediakan akses langsung untuk mengubah dan mengatur *forwarding plane* pada *network device* (*switch, router,* atau *access point*) baik *physical* maupun *virtual* (Foundation, 2012). Terdapat tiga bagian pada *switch* di dalam arsitektur SDN dengan protokol *OpenFlow*. Bagian tersebut adalah *Flow Table*, *Secure Channel*, dan *protokol OpenFlow* (Karakus & Durresi 2017). *OpenFlow channel* merupakan sebuah *interface* yang menghubungkan komunikasi antara *OpenFlow controller* dengan *OpenFlow switch*. Melalui *interface* tersebut, *controller* dapat mengatur konfigurasi dan manajemen dari *switch*, menerima laporan *event* dari *switch*, dan mengirimkan paket menuju *switch*. Dalam implementasinya, *controller* dapat mengirim pesan dengan format yang sesuai dengan *OpenFlow switch protocol*. *OpenFlow switch protocol* mendukung tiga jenis pesan (Foundation, 2014), yaitu:

- *Control-to-switch*: Pesan dikirim oleh *controller* menuju *switch* dan dapat membutuhkan atau tidak membutuhkan respon dari *switch*.
- *Asyncronous*: Pesan dikirim dari *switch* tanpa membutuhkan perintah dari *controller*. *Switch* mengirimkan pesan kepada *controller* untuk memberitahu kedatangan paket atau perubahan *state* pada *switch*.
- *Symmetric*: Pesan dikirim tanpa ada perintah baik dari kedua belah pihak.

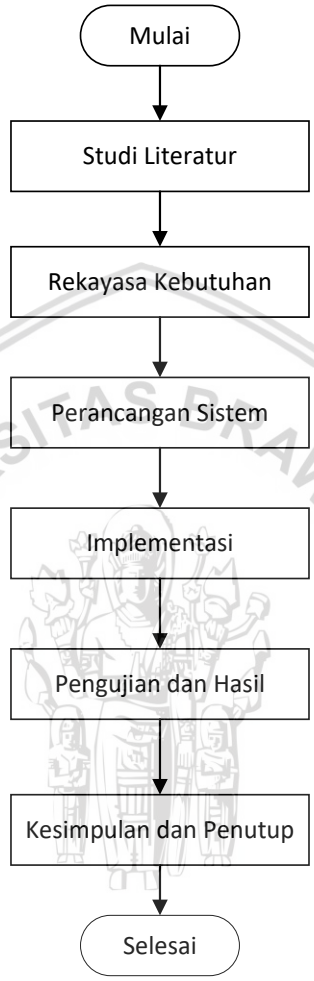
Protokol *OpenFlow* memiliki beberapa seri yang masing-masing memiliki karakteristik yang berbeda dari masing-masing serinya. Perbedaan disebabkan adanya penambahan fitur yang terjadi pada setiap pengulangan versi dari protokol *OpenFlow*. Tabel 2.2 menjelaskan daftar perbedaan yang terdapat pada setiap versi dari protokol *OpenFlow*.

Tabel 2.2 Perbedaan Setiap Versi *OpenFlow*

Versi	Fitur Utama
1.0 - 1.1	*Multiple Table* *Group Table* Mendukung penuh *VLAN* dan *MPLS*
1.1 - 1.2	*OXM Match* *Multiple Controller*
1.2 - 1.3	*Meter table* *Table miss entry*
1.3 - 1.4	*Synchronized Table* *Bundle*
1.4 - 1.5	*Egress Table* *Bundle* Terjadwal

BAB 3 METODOLOGI

Pada bab ini menjelaskan tentang metode dan langkah yang akan dilakukan dalam penelitian pengembangan *server failover* pada *Software Defined Network*. Diagram alir dari keseluruhan proses penelitian dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Alir Metodologi Penelitian

3.1 Studi Literatur

Studi literatur adalah sebuah kegiatan untuk melakukan pencarian informasi data dan pengumpulan bahan yang dapat digunakan untuk menunjang dalam penyusunan skripsi. Referensi tersebut berisikan tentang:

1. Jaringan *client server*
2. Mekanisme *fast-failover* pada *Software Defined Network* (SDN)
3. *Controller* menggunakan *Ryu*
4. Perancangan *Open vSwitch* sebagai *switch* pada komputer

5. Perhitungan *downtime*, *failback time*, dan *packet loss*. *Downtime* adalah waktu perpindahan dari *master server* menuju *slave server*. *Failback time* adalah waktu perpindahan dari *slave server* menuju *master server*. *Packet loss* adalah perhitungan data yang hilang pada saat pertukaran data terjadi.

Referensi ini dapat dicari dari buku, jurnal, artikel laporan penelitian, dan situs di internet. *Output* dari studi literatur ini adalah terkoleksinya referensi yang relevan dengan perumusan masalah. Tujuannya adalah untuk memperkuat permasalahan serta sebagai dasar teori dalam melakukan studi dan juga menjadi dasar untuk melakukan perancangan skenario pengujian. Hasil dari penelitian sebelumnya adalah untuk menguji *failover server* menggunakan parameter downtime dan *failback time*. Tetapi pada penelitian terdahulu peneliti tidak menyertakan pengujian *packet loss*. Sedangkan pada penelitian kali ini peneliti mengukur dengan parameter downtime, *failback time* dan *packet loss*.

Adapun sumber referensi utama adalah pada jurnal dan buku pada penelitian sebelumnya antara lain :

1. Penerapan Sistem *High Availability* Pada *Server* Teknik Informatika Universitas Brawijaya Menggunakan Aplikasi *Heartbeat* dan *Distributed Replicate Block Device* (DRBD) (Universitas Brawijaya Malang)
2. *High Availability Controller Software Defined Network* Menggunakan *Heartbeat* dan *DRBD* (Universitas Brawijaya Malang)
3. Implementasi *High Availability Server* Menggunakan Metode *Load Balancing* dan *Failover* Pada *Virtual Web Server Cluster* (Universitas Telkom)
4. Implementasi *High Availability Server* dengan teknik *Failover Virtual Computer Cluster* (Universitas Muhammadiyah Surakarta)
5. *High Availability Web Server* Menggunakan Protokol *Secure Shell* (Sekolah Tinggi Managemen dan Ilmu Komputer (STMIK) Bali)
6. Implementasi dan Analisa redundansi dan *High Availability* Dalam *Server* Untuk *Diskless Thin Client* Berbasis *Storage Area Network* (Universitas Indonesia)
7. Implementasi *Proxmox High Availability Server* Untuk Meningkatkan Kinerja *Website* Kampus (STMIK Raharja Tangerang dan STMIK Muhamamdiyah Banten)

3.2 Rekayasa Kebutuhan

Rekayasa kebutuhan ditujukan secara umum untuk melakukan analisis pada beberapa kebutuhan yang diperukan pada penelitian. Rekayasa kebutuhan pada penelitian ini meliputi:

1. Perangkat Keras (*Hardware*)
   a) PC/Laptop 5 buah

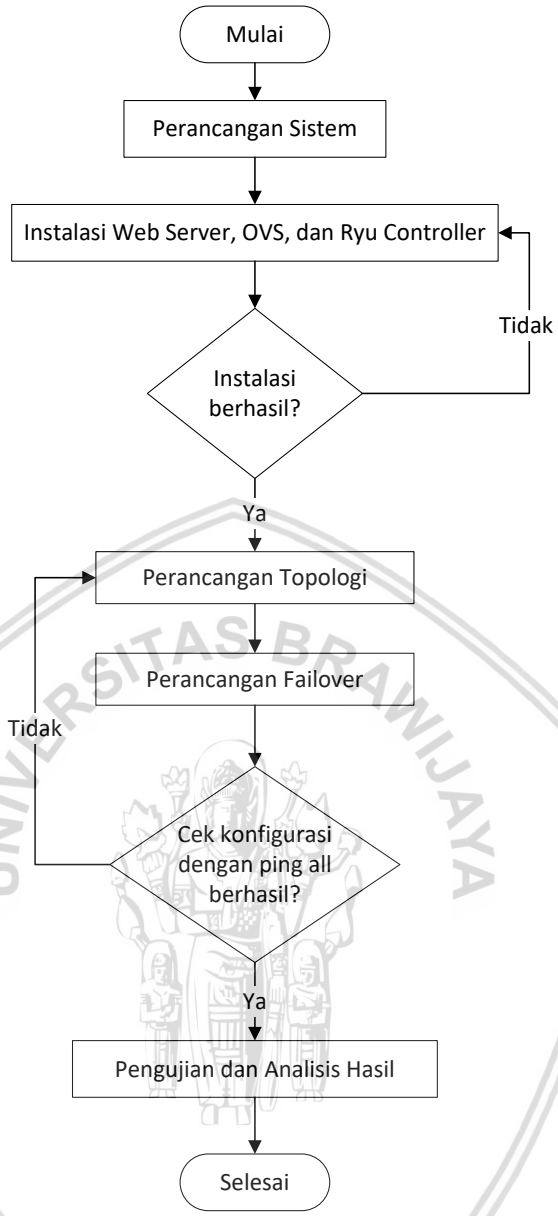
b) Kabel LAN 4 buah
c) USB Hub 1 buah
d) USB to LAN Port 3 buah

2. Perangkat Lunak (*Software*)
a) *Linux Ubuntu 14.04*
b) *Open vSwitch*
c) *Apache*
d) *Python*
e) *Ryu*
f) *Windows*

3.3 Perancangan Sistem

Perancangan sistem merupakan tahapan yang dilakukan untuk membangun sistem dari penelitian setelah melakukan studi literature dan analisis kebutuhan sistem. Tujuan dilakukan perancangan sistem agar implementasi sistem berjalan sistematis dan terstruktur.

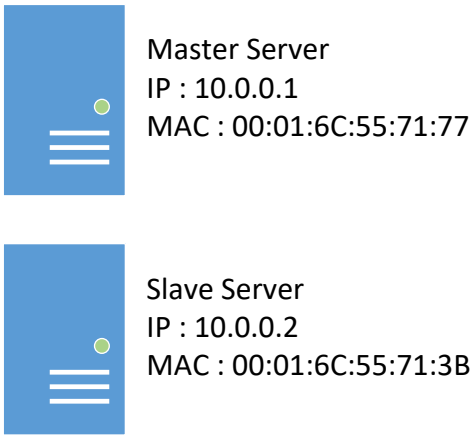
Berdasarkan gambar 3.2 dibawah merupakan alur pengerjaan sistem. Dimulai dengan perancangan sistem yang meliputi perancangan topologi dan perancangan *failover*. Penelitian ini menggunakan *mekanisme fast-failover*. Kemudian melakukan instalasi *apache* sebagai *web server*, *Open vSwitch* sebagai SDN *switch*, dan *Ryu* sebagai *controller*nya. Apabila proses instalasi telah selesai akan dilakukan proses pengecekan terlebih dahulu. Selanjutnya merancang topologi dan melakukan perancangan *failover* dengan membuat program *failover*. Program nantinya akan diimplementasikan pada *controller Ryu*. Setelah pengecekan konfigurasi dan sistem telah berjalan sesuai rancangan, berikutnya dilakukan pengujian dan analisis hasil.



Gambar 3.2 *Flowchart* Perancangan Umum Sistem

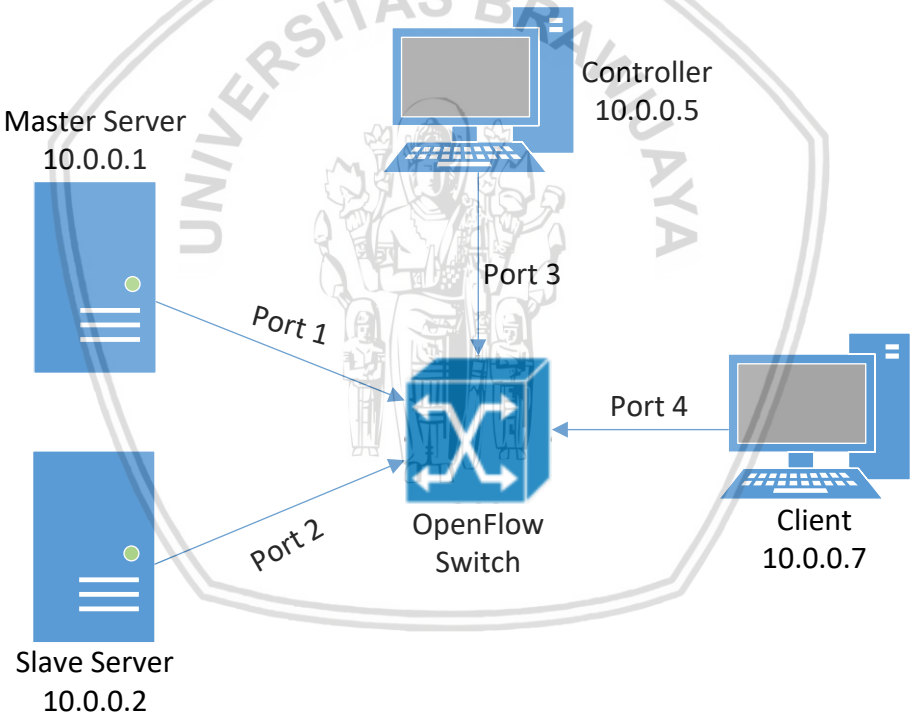
3.3.1 Perancangan *Server*

Penelitian ini mengimplementasikan dua buah *web server* yang akan digunakan untuk melayani *request* dari *client*. Dua buah *server* tersebut akan berjalan pada Sistem Operasi *Linux Ubuntu Server 14.04 LTS* dengan aplikasi berbasis *web* yang dibangun untuk menguji performa dari *server* dengan pengalamatan yang berbeda untuk *client* dan *server*. Pada gambar 3.3 menjelaskan sistem pengalamatan dari dua *server* tersebut.



Gambar 3.3 Alokasi Pengalamatan IP Pada *Server*

3.3.2 Perancangan Topologi



Gambar 3.4 Perancangan Topologi

Pada gambar 3.4 menjelaskan topologi jaringan yang digunakan pada penelitian ini, dimana akan mengimplementasikan arsitektur SDN dengan dua buah *server* yang telah terkonfigurasi dan siap untuk melayani *client*, satu buah SDN *controller* dan satu buah *switch* yang akan mengarahkan *request* dari *client* menuju *server*. *Switch* yang digunakan mengimplemntasikan *Open* v*Switch* pada komputer dengan Sistem Operasi *Linux Ubuntu 14.04 LTS*. Switch tersebut kemudian akan dikonfigurasi untuk meneruskan paket melalui 4 *port* yang

berbeda. *Port* pertama dengan *interface eth0* terhubung dengan *master server*, *port* kedua dengan *interface eth1* terhubung dengan *slave server*, port ketiga dengan *interface eth2* terhubung dengan *controller*, dan port terakhir dengan *interface eth3* terhubung dengan *client*. SDN *controller* dalam penelitian ini akan menggunakan sebuah komputer dengan Sistem Operasi *Linux Ubuntu Dekstop 14.04 LTS* dan sistem operasi jaringan. *Ryu 4.12* yang dapat diunduh dari github resmi *Ryu*. Protokol *OpenFlow 1.3* digunakan sebagai protokol komunikasi antara SDN *controller* dengan SDN *switch* dan *server*.

3.3.3 Perancangan *Failover*

Pada bagian ini peneliti menjelaskan perancangan failover yang dibuat oleh peneliti, dimana akan membuat program dengan Bahasa pemrograman *Python* yang digunakan untuk mengatasi masalah terhadap *server down*. Cara kerjanya dengan mengubah jalur pengiriman data dari *master server* menuju *slave server* pada *switch*.

3.3.4 Perancangan *Client*

Client digunakan untuk melakukan *request* terhadap *server*. *Client* akan menjalakan aplikasi untuk skenario pengujian. Fungsinya untuk mengetahui *downtime*, *failback time*, *packet loss*.

3.4 Implementasi

Implementasi rancangan yang telah dibuat dalam tahap perancangan penelitian. Dalam tahap ini dijelaskan proses implementasi dari kebutuhandan rancangan penelitian. Implementasi yang dilakukan antara lain implementasi *server*, implementasi arsitektur SDN, implementasi *server failover*, dan implementasi tools pengujian.

1. Implementasi *server*
   Implementasi pada tahap ini mencakup seluruh proses instalasi dan konfigurasi *server* serta aplikasi pendukung yang terdapat di dalam *server* seperti *apache*.
2. Implementasi *switch*
   Implementasi pada tahap ini mencakup seluruh proses instalasi dan konfigurasi menggunakan Sistem Operasi *Linux Ubuntu 14.04* dan aplikasi *Open vSwitch*.
3. Implementasi *controller*
   Implementasi pada tahap ini mencakup seluruh proses instalasi dan konfigurasi menggunakan Sistem Operasi *Linux Ubuntu 14.04* dan aplikasi yang mendukung modul *Ryu*. Serta menjalankan program *failover* pada *controller* agar program berjalan pada *switch*.
4. Implementasi *client*
   Implementasi program request packet pada *client* yang akan digunakan untuk menguji *server failover.*

3.5 Pengujian

Tahap pengujian pada penelitian ini dilakukan untuk menunjukkan bahwa sistem telah mampu bekerja sesuai spesifikasi yang mendasarinya. Penelitian ini bertujuan untuk menguji program *failover* dan kinerja *high availability server* yang telah dirancang. Untuk menguji hal tersebut, pengujian dibagi menjadi beberapa skenario pengujian, diantaranya:

3.5.1 Skenario Pengujian Fungsionalitas Program *Failover*

Dalam pengujian fungsionalitas program *failover* akan dilakukan dengan cara mengakses sebuah *website* pada *browser*. *Client* akan melakukan *request* ke *IP server* dengan perantara sebuah *controller* yang berperan sebagai *Virtual IP*. Virtual IP berfungsi untuk menginisialisasi IP *server* dan mengalihkan jalur komunikasi apabila *server* utama mati. Tujuannya yaitu untuk menjaga ketersediaan layanan *server*.

3.5.2 Skenario Pengujian Kinerja *High Availability Server*

Dalam menguji kinerja *high availability server*, pengujian terdiri dari tiga parameter yang diujikan, yaitu:

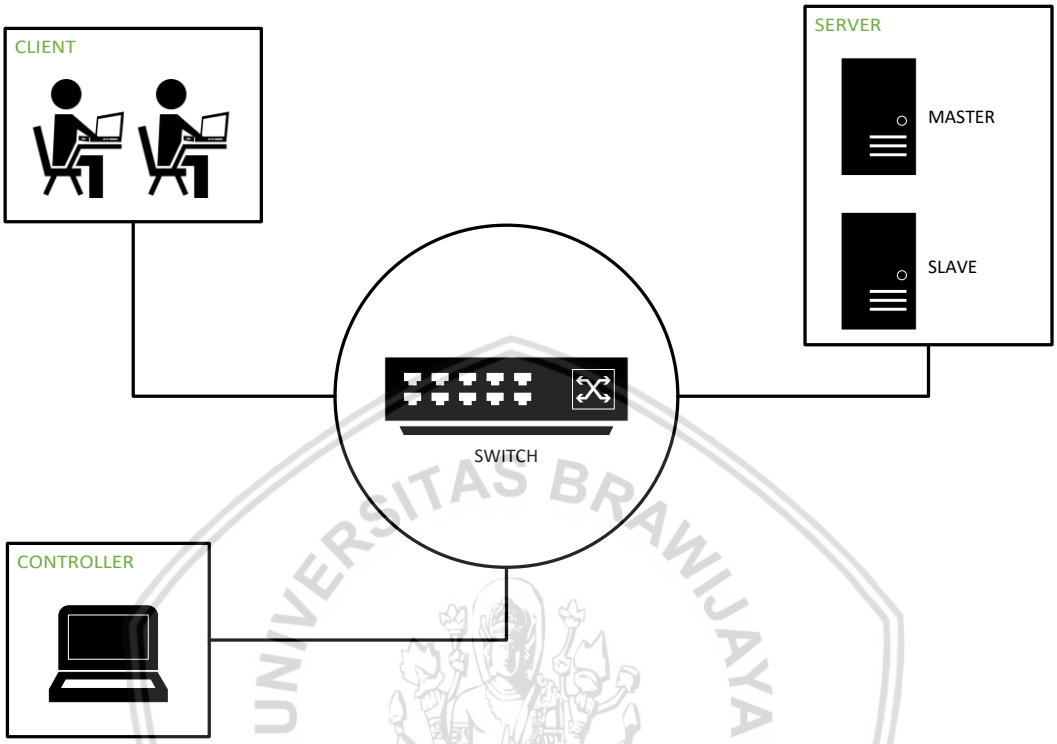
1. *Downtime*
   Dalam pengujian *downtime* akan dilakukan dengan cara melakukan akses terhadap *master server* lalu memutuskan jalur data *master server*. Kemudian diukur waktu perpindahan dari *master server* ke *slave server* menggunakan program pengiriman data secara terus menerus dari client ke server dengan cara waktu tiba paket pada *slave server* dikurangi waktu paket terakhir pada *master server*. Setelah dilakukan percobaan beberapa kali maka waktu tersebut akan dirata-rata agar mendapatkan hasil dari keseluruhan percobaan. Tujuannya untuk mengetahui waktu *failover*.
2. *Failback time*
   Dalam pengujian *failback* akan dilakukan dengan cara melakukan akses terhadap *slave server* lalu menyambungkan kembali jalur data *master server*. Kemudian diukur waktu perpindahan dari *slave server* ke *master server* menggunakan program pengiriman data secara terus menerus dari client ke server dengan cara waktu tiba paket pada *master server* dikurangi waktu paket terakhir pada *slave* server. Setelah dilakukan percobaan beberapa kali maka waktu tersebut akan dirata-rata agar mendapatkan hasil dari keseluruhan percobaan. Tujuannya untuk mengetahui waktu *failback*.
3. *Packet loss*
   Packet loss didefinisikan sebagai kegagalan transmisi paket IP mencapai tujuannya. Kegagalan paket mencapai tujuan dapat diakibatkan terjadinya *overload* trafik dalam jaringan, kongesti, *error* pada media fisik atau *overflow* yang terjadi pada *buffer* (Iskandar, 2015). Tujuannya untuk mengetahui tingkat keberhasilan pengiriman data.

3.6 Kesimpulan dan Saran

Pada tahap ini akan disimpulkan seluruh hasil dari analisa yang dilakukan sebelumnya. Kesimpulan diharapkan dapat menjawab keseluruhan dari rumusan masalah yang telah disampaikan sebelumnya. Saran akan diberikan untuk melengkapi penelitian di masa depan. Diharapkan penelitian ini dapat memberikan konstribusi bagi pengembang *failover* pada Arsitektur SDN.

BAB 4 REKAYASA KEBUTUHAN

4.1 Kebutuhan Sistem



Gambar 4.1 Gambaran Ruang Lingkup

Pada Gambar 4.1 terdapat 4 komponen yang digunakan yaitu *client*, *switch*, server, dan *controller*. Pada client digunakan untuk mengakses server, *controller* digunakan untuk menjalan program *failover*. Sedangkan diantara client, server, dan *controller* terdapat sebuah *switch* yang digunakan untuk menghubungkan ketiga komponen tersebut. Dimana pada masing-masing komponen memiliki kebutuhan perangkat keras dan perangkat lunak tersendiri. Dengan menyalakan *master server* dan *slave server* agar web *server* aktif. Alamat IP server yang digunakan yaitu 10.0.0.1 dan 10.0.0.2. Kemudian *controller* menjalankan sebuah program berbasis *Python*, yang didalamnya terdapat sebuah virtual IP. Virtual IP berfungsi untuk mentranslasikan alamat IP server yang diakses oleh client. Selain itu, fungsi virtual IP juga untuk menangani apabila *master server* mengalami down atau mati akan memindahkan jalur data yang terhubung ke *master server* menuju ke *slave server* (*failover*) dan juga sebaliknya apabila *master server* hidup kembali maka jalur data akan berpindah kembali menuju *master server* (*failback*). Alamat IP yang digunakan virtual IP yaitu 10.0.0.100. Ketika client mengakses sebuah server, virtual IP yang menghandel request client kemudian diteruskan menuju server. Setelah request client diterima oleh server, lalu server membalas request dengan mengirimkan paket yang diterima oleh virtual IP yang

kemudian diteruskan menuju client. Sehingga *controller* dapat mengontrol pertukaran data antara client dengan server menggunakan virtual IP.

4.1.1 Kebutuhan Perangkat Keras

Perangkat keras yang digunakan dalam sistem ini adalah sebuah PC dan laptop. Seperti pada tabel 4.1.

Tabel 4.1 Spesifikasi Perangkat Keras

Peran	Hardware	Spesifikasi
Master Server	Personal Komputer	Lenovo 3000 H Series Intel Dual Core E2220 / 2.4 Ghz 2GB RAM 320 GB SATA 7200 RPM
Slave Server	Personal Komputer	Lenovo 3000 H Series Intel Dual Core E2220 / 2.4 Ghz 2GB RAM 320 GB SATA 7200 RPM
Switch	Personal Komputer	Lenovo 3000 H Series Intel Dual Core E2220 / 2.4 Ghz 2GB RAM 320 GB SATA 7200 RPM
Controller	Laptop	ASUS A455L Intel Core i3-4030U / 1.90 Ghz 6GB RAM 120 GB SSD
Client	Laptop	MSI GL62MVR Intel Core i7-7700HQ / 2,8 Ghz up to 3.8 Ghz 8GB RAM 1 TB SATA 5400 RPM

4.1.2 Kebutuhan Perangkat Lunak

Perangkat lunak yang digunakan dalam sistem ini adalah sebagai berikut:

1. Linux Ubuntu

   Dalam penelitian ini menggunakan Sistem Operasi *Linux Ubuntu*. Dalam sistem operasi tersebut dapat diinstall aplikasi tertentu sesuai yang dibutuhkan. Sistem operasi ini digunakan untuk membuat *server* dan *client* agar sebuah sistem penelitian dapat tercipta. Untuk *server* menggunakan *Ubuntu Server* versi 14.04 LTS dan *client* menggunakan versi sama hanya saja versi desktopnya. Untuk *server* nantinya tambahan aplikasi yang akan dinstall antara lain *Apache, Python,* dan *Iperf* sedangkan *client* aplikasi yang dibutuhkan adalah *Python* dan *Iperf*.

2. Open v*Switch* 2.8.1

   Dalam penelitian ini menggunakan aplikasi *Open vSwitch 2.8.1*. Dalam aplikasi tersebut digunakan untuk membuat s*witch* yang dinstall pada Sistem Operasi *Linux Ubuntu* dan dipasang pada sebuah personal komputer. Fungsi dari *Open vSwitch* adalah membuat sebuah virtual *switch* pada sebuah komputer dengan menambah sebuah *port* LAN yang dipasang pada *port USB*. Penambahan *port* LAN tersebut dalam penelitian ini untuk mengimplementasikan yang sudah dirancang dalam sebuah topologi *Open vSwitch*. Kemudian pengaturan semua port tersebut disatukan menggunakan *mode bridge* dengan pengisian alamat IP satu saja, sehingga *Open vSwitch* dapat berjalan dengan baik.

3. OpenFlow 1.3

   Protokol *OpenFlow* sudah dijelaskan lengkap di bab 2 pada dasar teori. Dalam penelitian ini menggunakan *OpenFlow 1.3* karena *OpenFlow* versi ini telah mendukung fitur *fast failover.* Fitur ini berfungsi untuk menanggulangi adanya kesalahan dalam *server* dengan melihat *port* dan alamat IP yang aktif yang tergabung dalam sebuah *grup action*. Pemasangan *OpenFlow 1.3* ini dengan cara mengatur *controller* pada *switch* sehingga *controller* nanti dapat bekerja ketika terhubung pada *switch*. Apabila *controller* tidak dipasang maka suatu program yang dijalankan pada *controller* tidak dapat bekerja.

4. Apache

   Dalam penelitian ini menggunakan *Apache*. *Apache* adalah suatu sistem yang digunakan untuk membuat suatu *server* web. *Apache* tersebut diinstall pada sebuah komputer dengan Sistem Operasi *Linux Ubuntu Server*. Protokol yang digunakan untuk melayani fasilitas web ini menggunakan HTTP. Pada penelitian ini *server* yang digunakan berjumlah 2 buah. Setiap *server* telah diinstall *Apache* kemudian ditambahkan sebuah halaman website berbentuk html yang memiliki tampilan berbeda pada setiap *server*nya agar nantinya pada saat dilakukan pengujian dapat diketahui berjalan atau tidak.

5. Iperf

   Dalam penelitian ini menggunakan *Iperf*. *Iperf* adalah salah satu tool untuk mengukur *troughput bandwidth* dalam sebuah jalur jaringan. Agar bisa dilakukan pengukuran diperlukan Iperf yang terinstall *point to point*, baik disisi *server* maupun *client*. Iperf juga dapat digunakan untuk mengukur performa jalur dari sisi TCP maupun UDP. Pemasangan pada ubuntu dapat menggunakan perintah *apt-get install iperf*. Tool ini digunakan untuk mengetahui *packet loss* pada saat pengujian dalam penelitian ini.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

5.1 Perancangan

Pada perancangan kali ini peneliti akan menjabarkan apa saja yang dibutuhkan dalam merancang topologi seperti yang digambarkan pada Gambar 3.4, dimana pada perancangan tersebut terdiri dari perancangan *server*, perancangan *switch*, perancangan *controller*, dan perancangan *client*.

5.1.1 Perancangan *Server*

Server yang digunakan memilki spesifikasi sebagai berikut:

Tabel 5.1 Spesifikasi *Server*

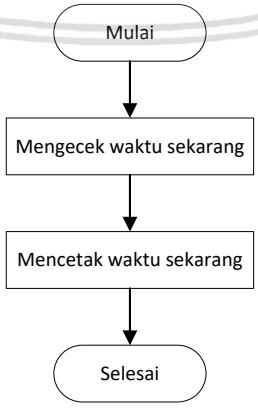
Master Server	***Slave Server***
Sistem Operasi : Ubuntu *Server* 14.04 LTS x64 Swap size : 8GB	**Sistem Operasi :** Ubuntu *Server* 14.04 LTS x64 Swap size : 8GB

Server akan berfungsi sebagai web *server* dengan menggunakan aplikasi apache sebagai dasarnya. Setelah melakukan instalasi dari *server* kemudian akan dilakukan konfigurasi pengalamatan IP dan MAC.

Tabel 5.2 Alokasi Pengalamatan *Server*

Server	**IP Address**	**MAC Address**
Master server	10.0.0.1	00:01:6C:55:71:77
Slave server	10.0.0.2	00:01:6C:55:71:3B

Pada tabel 5.2 menjelaskan pengalamatan IP untuk masing-masing *server*. Dimana *master server* menggunakan alamat IP 10.0.0.1 dan *slave server* menggunakan alamat IP 10.0.0.2.



Gambar 5.1 Flowchart Menampilkan Waktu Paket

Gambar 5.1 merupakan *flowchart* sebuah program dengan menggunakan bahasa pemrograman *Python* untuk mencetak waktu paket yang dikirim *client*. Program tersebut berfungsi untuk mengukur paket yang telah sampai *server* yang dikirmkan terus-menerus oleh *client*.

5.1.2 Perancangan *Switch*

Sebuah arsitektur SDN membutuhkan minimal sebuah *switch* yang mendukung protokol *OpenFlow*. *Switch* diimplementasikan dengan menggunakan *Open vSwitch* dan *controller* menggunakan *Ryu*. Pada penelitian ini *switch* yang digunakan memiliki spesifikasi seperti pada tabel 5.3.

Tabel 5.3 Spesifikasi *Switch*

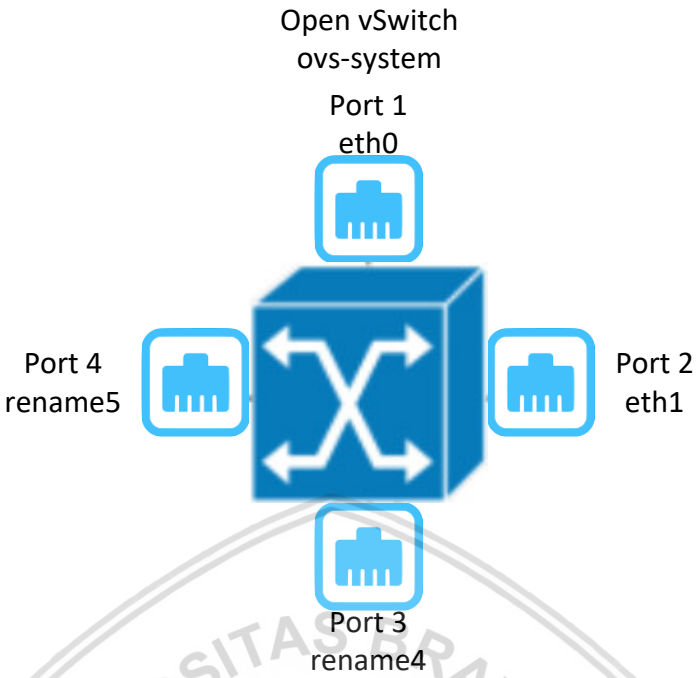
Switch		
Hardware	**Sistem Operasi**	**IP Address**
Lenovo 3000 H Series Intel Dual Core E2220 / 2.4 Ghz 2GB RAM 320 GB SATA 7200 RPM	Ubuntu *Server* 14.04 LTS x64 Swap size : 8GB	10.0.0.6

Open vSwitch adalah sebuah aplikasi yang digunakan untuk membentuk sebuah virtual *switch* pada Sistem Operasi *Linux*. *Open vSwitch* dapat diimplementasikan pada komputer desktop pada umumnya. Hal ini memberikan keuntungan dan fleksibilitas untuk mengembangkan kemampuan dari SDN *switch*. Pada penelitian ini *switch* dikembangkan pada sebuah komputer dengan Sistem Operasi *Linux Ubuntu* dan menggunakan aplikasi *Open vSwitch 2.8.1*. Komputer memiliki 4 buah *network interface* yang akan digunakan sebagai *port* dari *Open vSwitch* yang digunakan dalam penelitian ini. Seperti pada tabel 5.4.

Tabel 5.4 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*

Masing-masing port bertanggung jawab meneruskan paket dengan aturan yang terinstall pada flow-table. Gambar 5.2 menjelaskan topologi dari *Open vSwitch*.



Gambar 5.2 Topologi *Open vSwitch*

5.1.3 Perancangan *Controller*

Controller yang digunakan mempunyai spesifikasi seperti pada tabel 5.5.

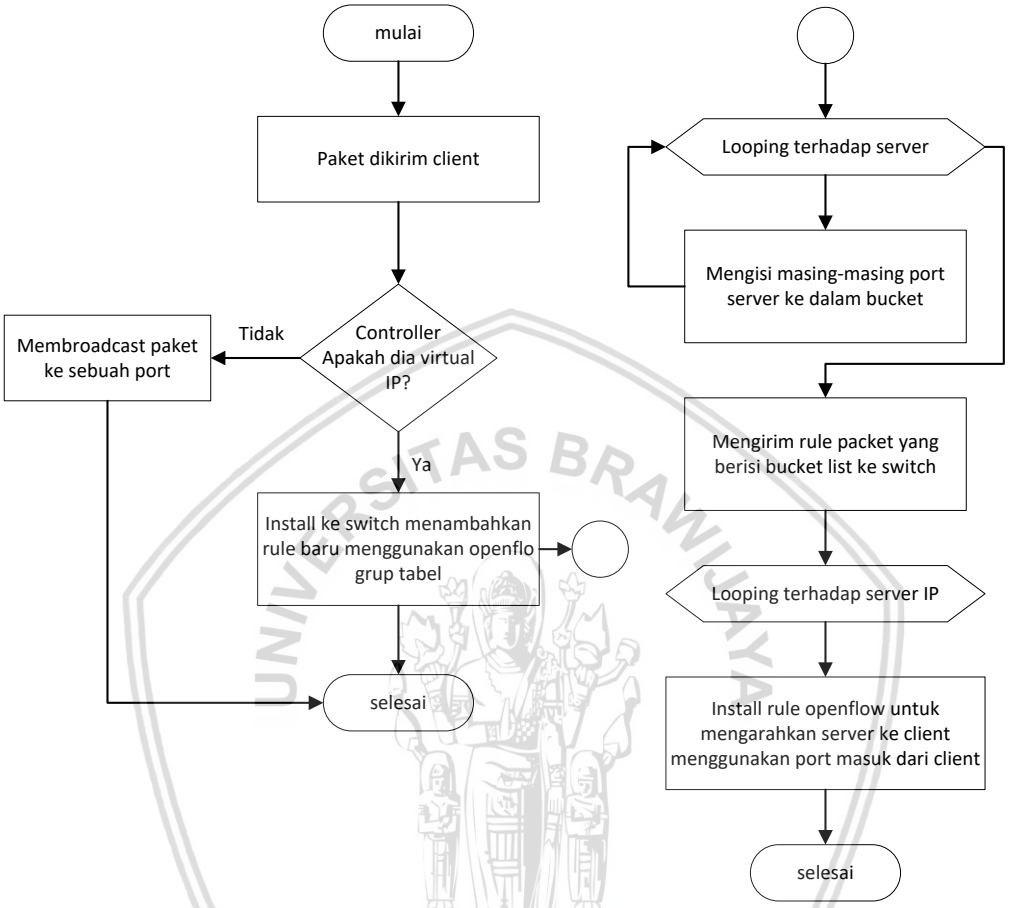
Tabel 5.5 Spesifikasi *Controller*

Controller		
Hardware	**Sistem Operasi**	**IP Address**
ASUS A455L Intel Core i3-4030U / 1.90 Ghz 6GB RAM 120 GB SSD	Ubuntu *Server* 14.04 LTS x64 Swap size : 4GB	10.0.0.5

Ryu merupakan sebuah *Network Operating System* yang didasarkan pada Bahasa pemrograman *Python*. *Ryu* memungkinkan untuk menuliskan aplikasi berbasis SDN menggunakan *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*. *Ryu* dapat diperoleh melalui github atau *packet installer* pada Sistem Operasi *Linux* Ubuntu.

5.1.3.1 Perancangan *Failover*

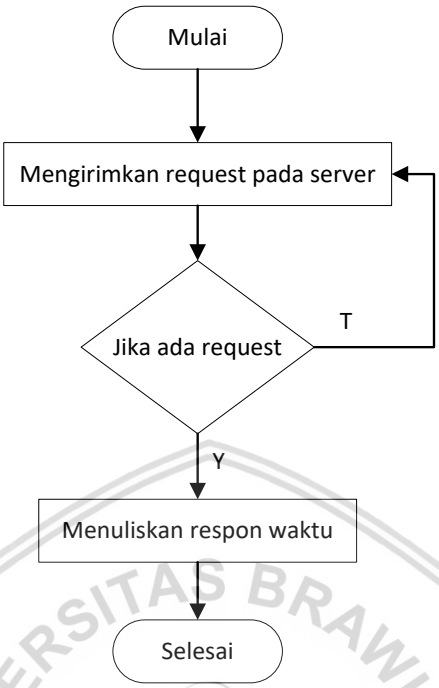
Failover berdasarkan status *server* yang bekerja diatas aplikasi *Ryu*. Modul untuk melakukan pengecekan terhadap *server* dijalankan oleh sebuah program *failover*. Berikut adalah alur flowchart program *failover*.



Gambar 5.3 Flowchart Program Failover

Dalam gambar 5.3 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.

5.1.4 Perancangan *Client*



Gambar 5.4 Flowchart Request Paket

Berdasarkan Gambar 5.4 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.

5.2 Implementasi

Pada bagian ini peneliti akan menjelaskan 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*.

5.2.1 Implementasi *Server*

Setelah melakukan proses instalasi *server* terlebih dahulu dilakukan konfigurasi pengalamatan IP dan MAC address dari masing-masing *server*. Pada Sistem Operasi *Linux* Ubuntu pengaturan IP address berada pada /etc/network/interface. Perintah yang digunakan adalah sebagai berikut.

```
$ sudo nano /etc/network/interface
```

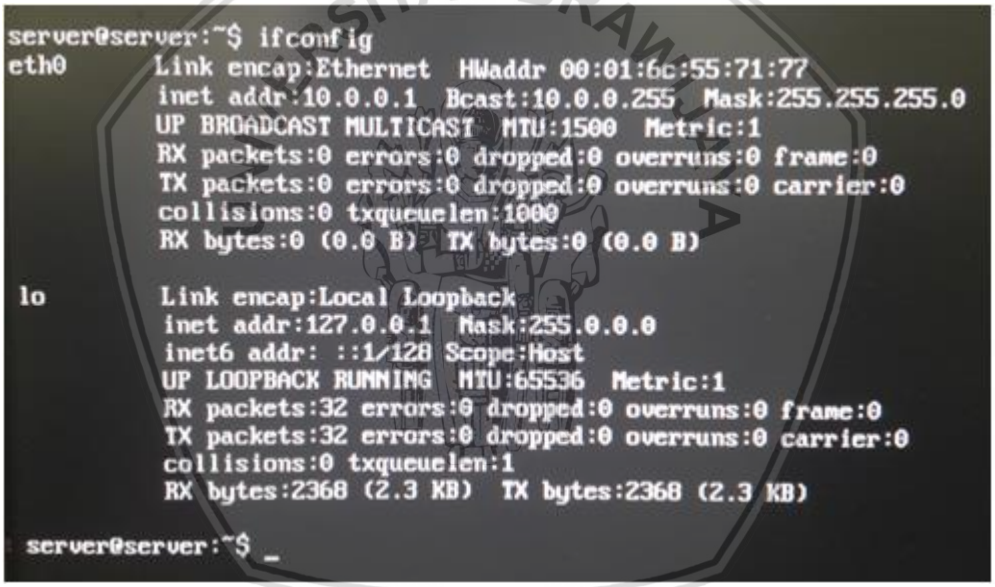
Kemudian konfigurasi yang digunakan adalah sebagai berikut.

```
auto eth0
iface eth0 inet static
   address 10.0.0.1
   netmask 255.255.255.0
   broadcast 10.0.0.255
   network 10.0.0.0
```

Untuk mengetahui konfigurasi telah berjalan dapat menggunakan perintah berikut.

```
$ sudo ifconfig
```

Perintah tersebut akan menampilkan konfigurasi dari interface. Proses tersebut kemudian dilakukan pada masing-masing *server* sesuai skema pengalamatan. Gambar 5.5 menunjukkan tampilan konfigurasi pada *server*.



Gambar 5.5 Konfigurasi IP pada *Server*

Untuk instalasi web *server* dibutuhkan Sistem Operasi *Linux Ubuntu* versi terbaru. Dengan melakukan pembaruan terlebih caranya adalah ketik perintah dibawah ini.

```
$ sudo apt-get update
```

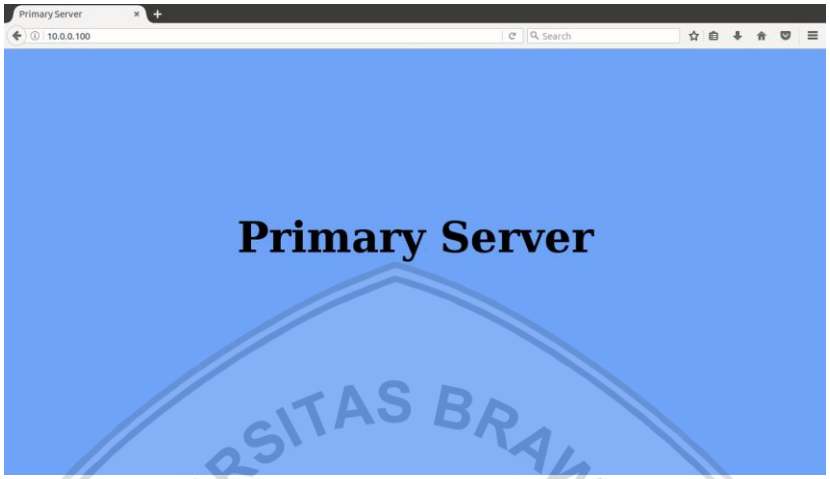
Setelah Sistem Operasi *Linux Ubuntu* telah diperbarui. Langkah selanjutnya adalah menginstal aplikasi yang digunakan untuk web *server* yaitu apache2. Perintah sebagai berikut.

```
$ sudo apt-get install apache2
```

Setalah terinstall apache2 dimasukkan sebuah website yang sudah didesain ke dalam directory apache. Perintah sebagai berikut.

```
$ cd /var/www/html
```

Setelah sebuah website sudah terpasang pada sebuah web *server* maka tampilan akan menjadi seperti dibawah.



Gambar 5.6 Tampilan Website *Master Server*

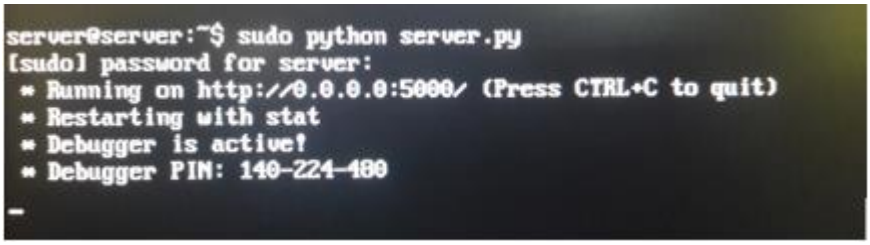
Pada gambar 5.6 adalah tampilan website *master server*. Dimana nantinya *client* mengakses *master server* akan menampilkan halaman depan website berwarna biru dengan tulisan *Primary Server*. Agar dapat mengetahui program *failover* bekerja atau tidak maka tampilan website pada *slave server* berbeda dengan *master server*. Website *slave server* mempunyai warna hijau muda dengan tulisan *Secondary Server*. Gambar 5.7 menunjukkan tampilan *slave server*.

Gambar 5.7 Tampilan Website *Slave Server*

Untuk menjalankan program request paket dari *client* menuju ke *server*. Program yang pada *server* tersebut dijalankan terlebih dahulu dengan perintah sebagai berikut:

```
$ sudo python server.py
```

Setelah perintah tersebut dijalankan maka program akan berjalan seperti pada gambar 5.8.



Gambar 5.8 Program pengiriman data telah aktif pada *server*

5.2.2 Implementasi *Switch*

Implementasi *switch* dilakukan dengan menggunakan sebuah *switch* yang mendukung protokol *OpenFlow* dan SDN *controller*. Dalam penelitian ini, sebuah komputer digunakan sebagai virtual *switch* yang akan bekerja sebagai *OpenFlow Switch*. Untuk dapat melakukan hal tersebut dibutuhkan sebuah aplikasi yang bernama *Open vSwitch* yang akan membuat sebuah virtual *switch* dan membuat topologi dari jaringan yang diinginkan. Kemudian akan dihubungkan dengan sebuah *controller* yang terhubung ke salah satu port di dalam *switch*. *Controller* yang digunakan dalam penelitian ini adalah Ryu. Untuk menginstall *Open vSwitch* perintah yang digunakan adalah sebagai berikut.

```
$ sudo apt-get install openvswitch
```

Setelah *Open vSwitch* terinstall dengan benar maka bridge br0 dapat ditambahkan. Bridge tersebut akan bekerja sebagai virtual *switch* yang akan meneruskan paket berdasarkan *flow-table* yang diberikan. Perintah yang dilakukan adalah sebagai berikut.

```
$ sudo ovs-vsctl add-br br0
```

Setelah bridge selesai ditambahkan, dapat dilakukan konfigurasi versi *OpenFlow* yang digunakan serta menambahkan *controller* pada *switch*. *Controller* ditambahkan dengan cara memberikan alamat ip dari *controller* serta protokol yang digunakan untuk komunikasi. Secara default port yang digunakan oleh *controller* adalah tcp:6633 sedangkan *switch* menggunakan port tcp:6634. Perintah yang dijalankan untuk melakukan kedua hal tersebut adalah sebagai berikut.

```
$ sudo ovs-vsctl set bridge br0 protocols=OpenFlow13
$ sudo ovs-vsctl set-server br0 tcp:10.0.0.5:6633
```

Setelah bridge selesai dikonfigurasi, dapat ditambahkan port yang digunakan oleh *switch* untuk meneruskan paket. Port tersebut harus memiliki sebauh interface yang terikat dengan port tersebut. Interface tersebut akan menerima dan meruskan paket yang dari host menuju *Open vSwitch* serta sebaliknya. Namun sebelum dapat menambahkan port, terlebih dahulu dilakukan konfigurasi

terhadap *interface* yang akan digunakan. Perintah yang digunakan untuk melakukan konfigurasi adalah sebagai berikut.

```
$ sudo nano /etc/network/interface
```

Kemudian konfigurasi yang digunakan adalah seperti berikut.

```
auto eth0
iface eth0 inet static
auto eth1
iface eth1 inet static
auto rename4
iface rename4 inet static
auto rename5
iface rename5 inet static
auto br0
iface br0
    address 10.0.0.6
    netmask 255.255.255.0
    network 10.0.0.0
    broadcast 10.0.0.255
```

Pada konfigurasi diatas, keempat interface (eth0, eth1, rename4, dan rename5) tidak memiliki IP address. Sedangkan br0 diberikan IP address 10.0.0.6. setelah konfigurasi dilakukan interface tersebut kemudian akan disambungkan ke bridge yang telah ditambahkan sebelumnya. Perintah yang digunakan adalah sebagai berikut.

```
$ sudo su
$ ip link set eth0 up
$ ip link set eth1 up
$ ip link set rename4 up
$ ip link set rename5 up
$ ovs-vsctl add-port br0 eth0
$ ovs-vsctl add-port br0 eth1
$ ovs-vsctl add-port br0 rename4
$ ovs-vsctl add-port br0 rename5
```

Setelah port ditambahkan, cek konfigurasi Open v *Switch* dengan perintah berikut.

```
$ sudo ovs-vsctl show
```

Perintah tersebut akan menampilkan konfigurasi seperti pada gambar 5.9

Gambar 5.9 Konfigurasi Open v*Switch*

Perintah gambar tersebut terdapat alamat *controller*, namun *controller* masih belum terkoneksi ke dalam sistem. Apabila *controller* telah terkoneksi, perintah yang sama akan menampilkam konfigurasi seperti pada gambar 5.10.

Gambar 5.10 Konfigurasi Open v*Switch* Terhubung *Controller*

5.2.3 Implementasi *Controller*

Setelah *Open vSwitch* terkonfigurasi dan berjalan dengan lancar, dapat dilakukan instalasi *Ryu* pada computer *controller*. Proses instalasi dari *Ryu* dapet dilakukan menggunakan dua cara, melalui apt-get atau melalui kompilasi langsung dari source code. Berikut adalah perintah yang dijalankan untuk melakukan instalasi *Ryu* dari source code.

```
$ sudo apt-get install git
$ git clone git://github.com/ryu.git
$ cd ryu/
$ python ./setup.py install
```

Setelah instalasi selesai perlu dilakukan konfigurasi pengalamatan IP agar *controller* terhubung dengan *Open vSwitch*. Perintah yang dilakukan sebagai berikut.

```
$ sudo nano /etc/network/interfaces
```

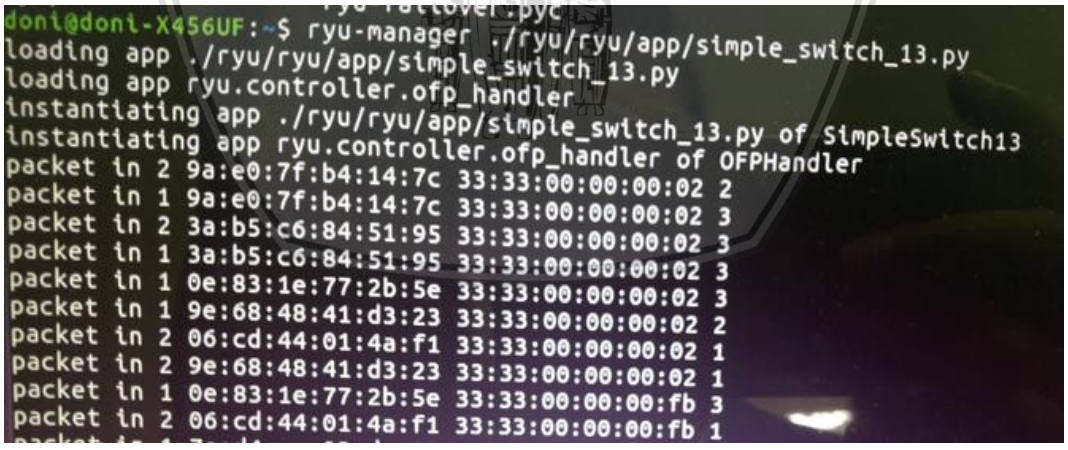
Kemudian konfigurasi yang diterapkan adalah sebagai berikut.

```
auto eth0
iface eth0 inet static
    address 10.0.0.5
    netmask 255.255.255.0
    gateway 10.0.0.6
    broadcast 10.0.0.255
    network 10.0.0.0
```

Setelah konfigurasi selesai dapat dilakukan percobaan dengan menjalakan aplikasi simple *switch* bawaan dari *Ryu*. Perintah yang dijalankan seperti berikut.

```
$ ryu-manager simple_switch_13.py
```

Apabila proses instalasi berjalan dengan benar maka akan terlihat tampilan seperti pada gambar 5.11.



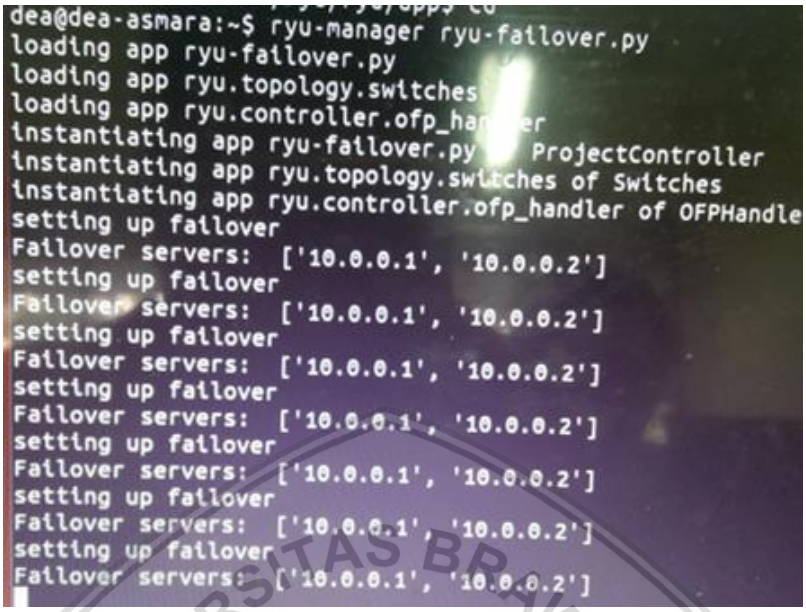
Gambar 5.11 Interface ryu-manager

5.2.3.1 Implementasi *Failover*

Failover berjalan pada aplikasi *Ryu* yang dijalankan pada *controller*. Untuk menjalankan aplikasi *failover*, masukkan perintah berikut.

```
$ sudo ryu-manager ryu-failover.py
```

Setelah aplikasi berjalan maka console akan menampilkan seperti gambar 5.12.



Gambar 5.12 Aplikasi failover

Dalam program *failover* bekerja diatas aplikasi *Ryu*, method yang dituliskan adalah `def failover_handler`. Dalam method tersebut dijelaskan bahwa yang dibutuhkan data dalam kinerjanya seperti IP, MAC, dan port. Data tersebut digunakan untuk menentukan *server* mana yang aktif dan port yang akan dilalui lalu dimasukkan ke dalam *bucket*. *Bucket* tersebut yang nantikan akan menjadi sebuah data utama agar program *failover* berkerja atau tidak yang dituliskan `buckets=[]`. Program *failover* juga membutuhkan perulangan dalam pencarian *server* dan *client*. Setelah *server* dan *client* didapatkan makan data *server* dan *client* disimpan di *switch* dalam bentuk tabel flow. Dengan melihat port aktif atau non aktif maka *failover* akan bekerja menggantikan port non aktif dengan port yang aktif. Serta datfar bucket yang aktif dan non aktif. Berikut program *failover* dibawah ini.

Kode Sumber Program Failover

```
    # inisialisasi sistem failover
1   def failover_handler(self, ev, src_ip, src_mac, in_port):
2       print "setting up failover"
3       msg = ev.msg
4       datapath = msg.datapath
5       ofp = datapath.ofproto
6       ofp_parser = datapath.ofproto_parser
7       buckets = []
8   # untuk menentukan server yang dituju client
```

```
for server_ip in self.server_ips:
    server_mac = self.arp_table[server_ip]
    outport = self.mac_to_port[datapath.id][server_mac]
    bucket_weight = 0
    bucket_action =
        [ofp_parser.OFPActionSetField(eth_dst=server_mac),
         ofp_parser.OFPActionSetField(ipv4_dst=server_ip),
         ofp_parser.OFPActionOutput(outport)]
    buckets.append(
         ofp_parser.OFPBucket(
             weight=bucket_weight,
              watch_port=outport,
              watch_group=ofp.OFPG_ANY,
              actions=bucket_action
        )
   )
group_id = random.randint(0, 2**32)
req = ofp_parser.OFPGroupMod(
    datapath, ofp.OFPGC_ADD, ofp.OFPGT_FF, group_id,
    buckets
)
datapath.send_msg(req)
group_action = [ofp_parser.OFPActionGroup(group_id)]
# menyimpan data informasi pada tabel flow entri
match = ofp_parser.OFPMatch(in_port=in_port,
        eth_type=ether_types.ETH_TYPE_IP,
        eth_src=src_mac, eth_dst=self.virtual_mac,
        ipv4_src=src_ip, ipv4_dst=self.virtual_ip)

self.add_flow(datapath, 1, match, group_action)
# mencari rute dari server ke client
for server_ip in self.server_ips:
    server_mac = self.arp_table[server_ip]
    outport = self.mac_to_port[datapath.id][server_mac]
    match = ofp_parser.OFPMatch(in_port=outport,
                          eth_type=ether_types.ETH_TYPE_IP,
                          eth_src=server_mac, eth_dst=src_mac,
```

```
46                              ipv4_src=server_ip,ipv4_dst=src_ip)
47      actions=
48  ([ofp_parser.OFPActionSetField(eth_src=self.virtual_mac),
49       ofp_parser.OFPActionSetField(ipv4_src=self.virtual_ip),
50       ofp_parser.OFPActionOutput(in_port)])
51      self.add_flow(datapath, 1, match, actions)
```

Berdasarkan penjelasanan program diatas:

1. Baris 1: inisialisai mendapatkan fungsi failover
2. Baris 3: melakukan pengiriman pesan
3. Baris 4: mendapatkan pesan data dari *switch*
4. Baris 9: pengecekan IP *server* yang sudah di set pada proram.
5. Baris 10: IP dan MAC tersebut dimasukkan ke dalam sebuah arp tabel. Kemudian *outport* berisi *port* keluar yang menuju ke *server*.
6. Baris 12: menginisialisai *bucket list* dengan nilai 0.
7. Baris 13: suatu *bucket_action* berisi alamat dan IP versi 4 tujuan yang di dalamnya berupa MAC *server* dan IP *server* serta *port* keluar yang menuju *server*.
8. Baris 17: semua data tersebut di masukkan dalam satu paket.
9. Baris 25: ID *group* yang diinisialisai secara random antara 0-2^{32}.
10. Baris 26: *controller* meminta *request* terhadap *switch* dengan isi *datapath*.
11. Baris 30: datapath mengirim request
12. Baris 31: melakukan balasan dari *switch* dari *request controller* dan kemudian grup *action* berisi id grup.
13. Baris 33: mendapatkan informasi data informasi berisi port masuk pada *switch*, tipe *port* yang digunakan adalah IP, *port* sumber berisi MAC , *port diambil dari MAC virtual*.
14. Baris 38: menambahkan pada tabel flow.
15. Baris 40: pengecekan IP *server* yang sudah di set pada proram.
16. Baris 43: IP dan MAC tersebut dimasukkan ke dalam sebuah arp tabel. Kemudian *outport* berisi port keluar menuju *client*.
17. Baris 47: mendapatkan informasi data informasi berisi port masuk pada *switch*, tipe *port* yang digunakan adalah IP, *port* sumber berisi MAC, *port diambil dari MAC virtual*.
18. Baris 52: menambahkan pada tabel flow.

5.2.4 Implementasi *Client*

Client menggunakan aplikasi Iperf untuk mengirimkan *request* kepada *server* yang digunakan untuk pengujian. Perintah yang dilakukan untuk menginstall adalah sebagai berikut.

```
$ sudo apt-get install iperf
```

Setelah proses instalasi selesai perlu dilakukan konfigurasi pengalamatan IP agar *client* terhubung dengan *Open Vswitch*. Perintah yang dilakukan sebagi berikut.

```
$ sudo nano /etc/network/interface
```

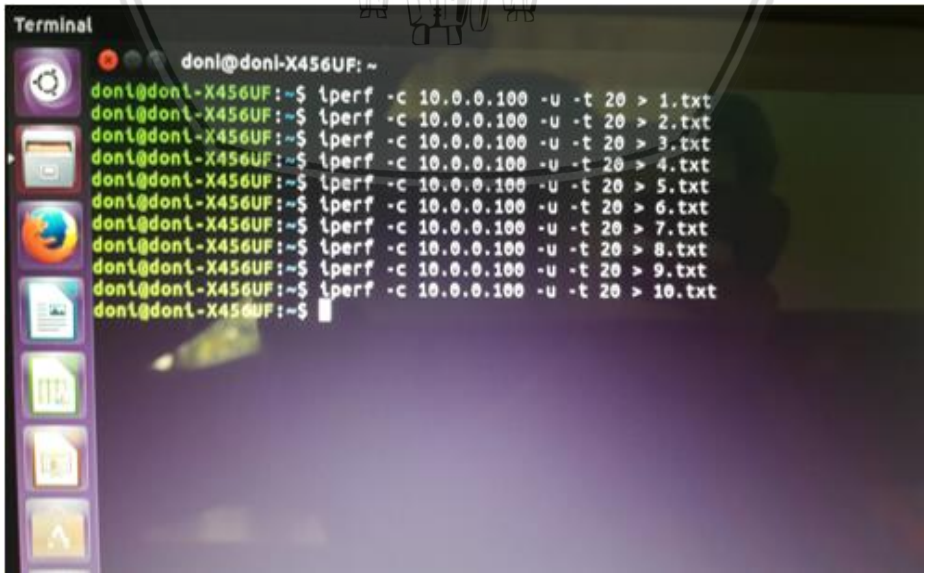
Kemudian konfigurasi yang diterapkan adalah sebagi berikut.

```
auto eth0
Iface etho inet static
   address 10.0.0.3
   netmask 255.255.255.0
   gateway 10.0.0.6
   broadcast 10.0.0.255
   network 10.0.0.0
```

Untuk menjalankan pengujian dengan mengirimkan paket selama 20 detik. Percobaan tersebut dilakukan sebanyak 10 kali menggunakan perintah sebagai berikut.

```
$ iperf -c 10.0.0.100 -u -t 20
```

Apabila aplikasi berjalan dengan benar maka *console* akan menampilkan seperti pada gambar 5.13.



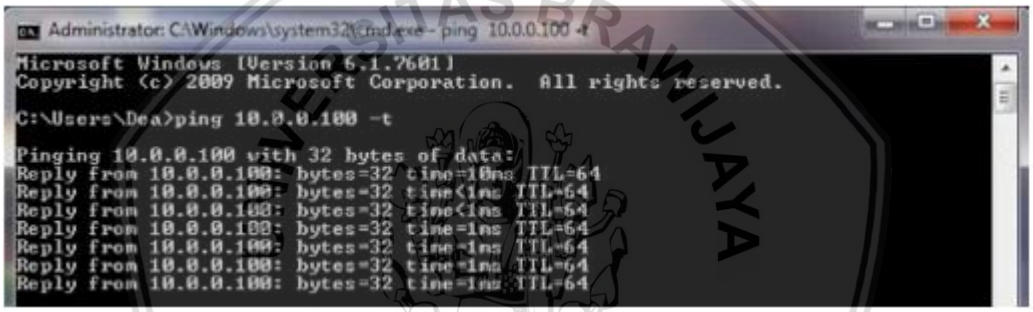
Gambar 5.13 Iperf Pada *Client*

BAB 6 PENGUJIAN DAN ANALISIS

Setelah proses perancangan dan implementasi dilakukan maka tahap selanjutnya pada penelitian ini adalah dengan melakukan pengujian pada sistem yang telah dibangun. Proses pengujian meliputi pengujian fungsionalitas dan pengujian kinerja. Pengujian fungsionalitas akan menguji rancangan sistem *high availability server* pada *Software Defined Network* apakah sudah berjalan sesuai perancangan yang telah dilakukan. Sedangkan pengujian kinerja high availability server meliputi pengujian *downtime*, pengujian *failback time*, dan *packet loss*.

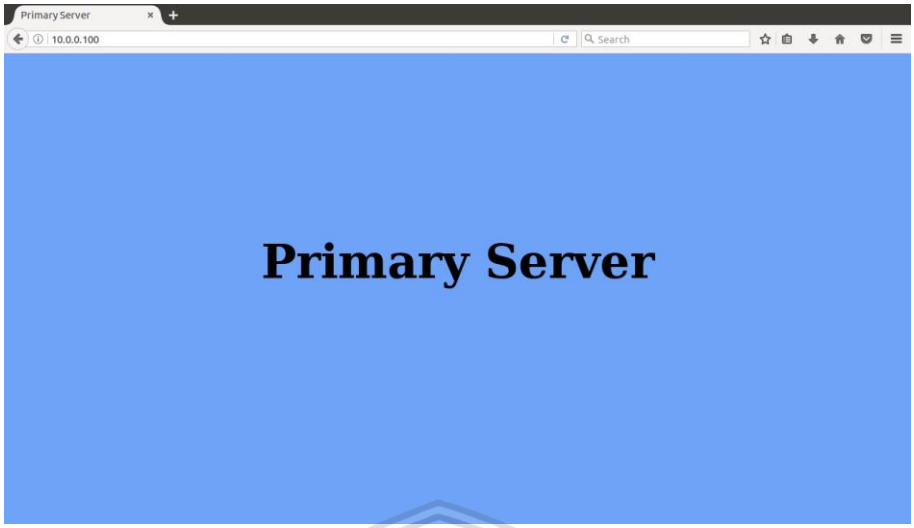
6.1 Pengujian Fungsionalitas Program Failover

Pada pengujian ini dilakukan *ping test* untuk mengetahui jaringan yang ada terkoneksi dengang baik, dilakukan pengiriman request dari *client* menuju *server* dengan melakukan ping terhadap virtual IP terlebih dahulu. Alamat virtual IP adalah 10.0.0.100. Gambar 6.1 menunjukan proses ping pada virtual IP.



Gambar 6.1 Ping Alamat Virtual IP

Dari gambar 6.1 terlihat bahwa *client* dan *server* sudah saling terhubung yang artinya *client* bisa melakukan request ke *server* dan *server* bisa mengenali *client* serta memberikan sevice apache ke *client* yang berguna untuk memfungsikan situs web. Setelah melakukan ping ke virtual IP, kemudian *client* membuka sebuah browser lalu mengakses alamat virtual IP dengan menuliskan IP 10.0.0.100. hasilnya virtual IP akan menampilkan website *master server*. Karena pada program *failover* telah diatur *master server* menjadi *server* utama terlebih dahulu. Gambar 6.2 menunjukkan tampilan *master server* pada saat diakses oleh *client* dengan menggunakan browser Google Chrome.

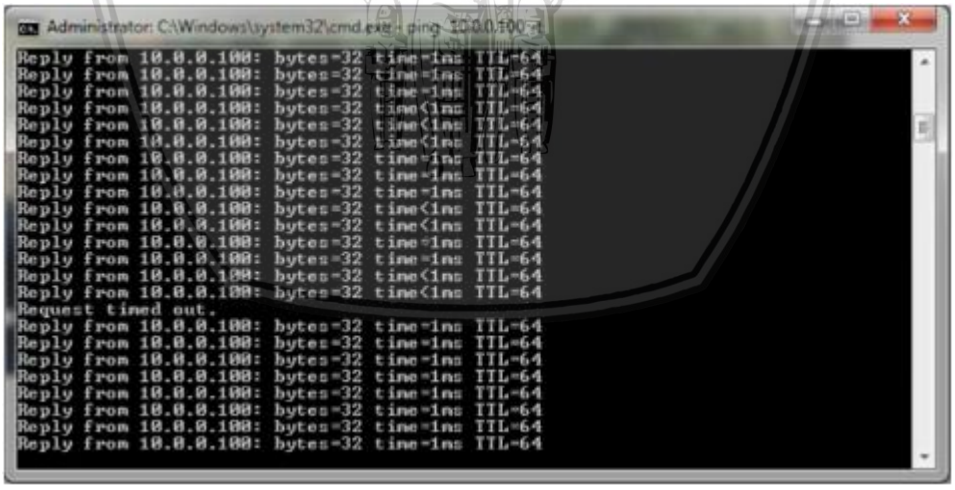


Gambar 6.2 Tampilan Website Pada Saat Mengakses *Master Server*

Selama *master server* menjadi *server* utama, semua *client* yang mengakses virtual IP akan diarahkan menuju *master server*. Semua request dari *client* akan dilayani oleh *master server*. Kemudian untuk mencoba program *failover* dilakukan pemutusan pada jalur menuju *master server* dengan cara mematikan port pada *switch* yang terhubung pada *master server*. Perintah untuk mematikan port *master server* pada *switch* sebagai berikut:

```
$ sudo ifconfig eth0 down
```

Hasilnya terlihat pada gambar 6.3.



Gambar 6.3 Menunjukkan saat jalur data pada *master server* terputus

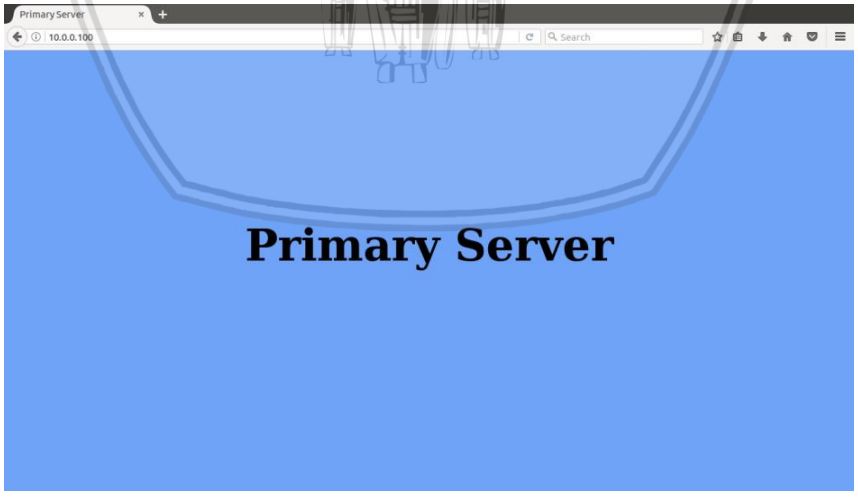
Pada saat terputusnya jalur data *master server* virtual IP mengarahkan jalur pengiriman data menuju *slave server*. Sehingga *slave server* menjadi *server* utama. Semua request *client* dari *master server* dilayani oleh *slave server*. Hasilnya dalam browser tampilan website juga berubah. Akan tetapi IP yang diakses oleh *client* tidak berubah (sama). Gambar 6.4 menunjukkan tampilan *slave server* pada saat diakses dengan menggunakan virtual IP.

Gambar 6.4 Tampilan Website Pada Saat Mengakses *Slave Server*

Pada saat *master server* mengalami jalur data down atau mati maka *controller* akan mengalihkan jalur komunikasi dari *master server* ke *slave server*. Kemudian pada saat jalur data *master server* tersambung kembali. Dengan menuliskan perintah pada *switch* sebagai berikut:

```
$ sudo ifconfig eth0 up
```

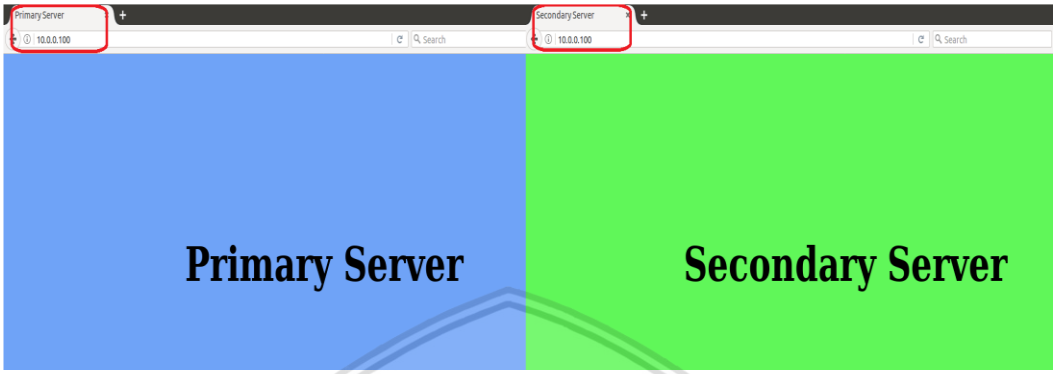
Setelah perintah tersebut berjalan, virtual IP akan mengarahkan kembali ke *master server*. *Master server* menjadi *server* utama kembali. Hasilnya semua request *client* yang tadi menuju ke *slave server* dikembalikan ke *master server*. Serta request *client* dilayani kembali oleh *master server*. Sehingga tampilan website pada browser akan berubah kembali ke *master server*. Gambar 6.5 menunjukkan tampilan website pada browser kembali ke *master server*.



Gambar 6.5 Tampilan Website Kembali Ke *Master Server*

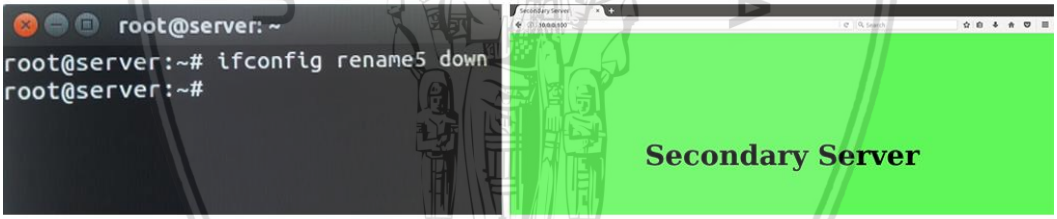
Pada saat *master server* mengalami jalur data up atau aktif maka *switch* akan mengalihkan jalur komunikasi dari *slave server* ke *master server*.

Hasil pengujian fungsionalitas didapatkan beberapa hasil yaitu hasil perpindahan website dari *master server* ke *slave server* begitupun sebalikan pada proses *failback*, keberhasilan dari pengujian di perlihatkan dari gambar 6.6 dibawah ini yang menunjukkan hasil perpindahan dari *master server* ke *slave server* tanpa merubah alamat ip.



Gambar 6.6 Tampilan website primary dan secondary dengan IP sama

Perubahan dari *master server* ke *slave server* diatur oleh *switch* yang didalamnya terdapat protokol *OpenFlow* dimana terdapat mekanisme *fast-failover* untuk mengatasi *failover server*, ini ditunjukkan dimana *client* bisa mengakses website meskipun *master server* (*server* utama) mati, ini terlihat dari gambar dibawah ini:



Gambar 6.7 Perintah down *master server* dan *client* akses web

Pada gambar 6.7 menunjukkan perintah untuk memutuskan jalur komunikasi dari *switch* ke *master server*, akan tetapi *client* masih bisa mengakses website. Ini memperlihatkan bahwa *switch* mengalihkan request *client* ke *slave server*. Pada proses *failback switch* akan mengalihkan request *client* kembali ke *master server* apabila *master server* up kembali dengan menggunakan mekanisme *fast-failover* didalam *controller*, hal ini dilakukan dengan melakukan up *master server* melalui *command line* dengan perintah :

Gambar 6.8 Perintah untuk mengaktifkan *master server*

Pada gambar 6.8 perintah tersebut maka *master server* akan up kembali dan *client* akan bisa mengakses website yang ada di *master server* kembali.

6.2 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*.

6.2.1 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*. Di bawah ini adalah tampilan log waktu pada *master server, slave server,* dan *client*.

Log *master server*

```
2018-03-19 14:49:01.238737
10.0.0.7 - - [19/Mar/2018 14:49:01] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:02.253625
10.0.0.7 - - [19/Mar/2018 14:49:02] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:03.266969
10.0.0.7 - - [19/Mar/2018 14:49:03] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:04.281723
10.0.0.7 - - [19/Mar/2018 14:49:04] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:05.296415
10.0.0.7 - - [19/Mar/2018 14:49:05] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:06.309428
10.0.0.7 - - [19/Mar/2018 14:49:06] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:07.322564
10.0.0.7 - - [19/Mar/2018 14:49:07] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:08.337064
10.0.0.7 - - [19/Mar/2018 14:49:08] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:09.350415
10.0.0.7 - - [19/Mar/2018 14:49:09] "GET / HTTP/1.1" 200 -
```

```
2018-03-19 14:49:10.365420
10.0.0.7 - - [19/Mar/2018 14:49:10] "GET / HTTP/1.1" 200 -
```

Log *slave server*

```
2018-03-19 14:49:10.386737
10.0.0.7 - - [19/Mar/2018 14:49:10] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:11.39975
10.0.0.7 - - [19/Mar/2018 14:49:11] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:12.414638
10.0.0.7 - - [19/Mar/2018 14:49:12] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:13.429643
10.0.0.7 - - [19/Mar/2018 14:49:13] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:14.444531
10.0.0.7 - - [19/Mar/2018 14:49:14] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:15.457544
10.0.0.7 - - [19/Mar/2018 14:49:15] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:16.472549
10.0.0.7 - - [19/Mar/2018 14:49:16] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:17.487437
10.0.0.7 - - [19/Mar/2018 14:49:17] "GET / HTTP/1.1" 200 -
2018-03-19 14:49: 18.50045
10.0.0.7 - - [19/Mar/2018 14:49:18] "GET / HTTP/1.1" 200 -
2018-03-19 14:49: 19.515455
10.0.0.7 - - [19/Mar/2018 14:49:19] "GET / HTTP/1.1" 200 -
```

Log *client*

```
2018-03-19 14:49:01.238737
10.0.0.7 - - [19/Mar/2018 14:49:01] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:02.253625
10.0.0.7 - - [19/Mar/2018 14:49:02] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:03.266969
10.0.0.7 - - [19/Mar/2018 14:49:03] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:04.281723
10.0.0.7 - - [19/Mar/2018 14:49:04] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:05.296415
10.0.0.7 - - [19/Mar/2018 14:49:05] "GET / HTTP/1.1" 200 -
```

```
2018-03-19 14:49:06.309428
10.0.0.7 - - [19/Mar/2018 14:49:06] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:07.322564
10.0.0.7 - - [19/Mar/2018 14:49:07] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:08.337064
10.0.0.7 - - [19/Mar/2018 14:49:08] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:09.350415
10.0.0.7 - - [19/Mar/2018 14:49:09] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:10.386737
10.0.0.7 - - [19/Mar/2018 14:49:10] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:11.39975
10.0.0.7 - - [19/Mar/2018 14:49:11] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:12.414638
10.0.0.7 - - [19/Mar/2018 14:49:12] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:13.429643
10.0.0.7 - - [19/Mar/2018 14:49:13] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:14.444531
10.0.0.7 - - [19/Mar/2018 14:49:14] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:15.457544
10.0.0.7 - - [19/Mar/2018 14:49:15] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:16.472549
10.0.0.7 - - [19/Mar/2018 14:49:16] "GET / HTTP/1.1" 200 -
2018-03-19 14:49:17.487437
10.0.0.7 - - [19/Mar/2018 14:49:17] "GET / HTTP/1.1" 200 -
2018-03-19 14:49: 18.50045
10.0.0.7 - - [19/Mar/2018 14:49:18] "GET / HTTP/1.1" 200 -
2018-03-19 14:49: 19.515455
10.0.0.7 - - [19/Mar/2018 14:49:19] "GET / HTTP/1.1" 200 -
```

Berdasarkan tampilan log waktu diatas menunjukkan bahwa *client* masih bisa mengakses *server* meskipun *master server down*. Pada Tabel 6.1 dibawah ini merupakan hasil pengujian *downtime* yang didapat setelah dilakukan 10 kali percobaan.

Tabel 6.1 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

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 hasil rata-rata waktu downtime yang didapat dari 10 kali percobaan didapatkan waktu selama 0,826 detik.

6.2.2 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*. Di bawah ini adalah tampilan log waktu pada *slave server, master server,* dan *client*.

Log *slave server*

```
2018-03-19 16:30:01.109878
10.0.0.7 - - [19/Mar/2018 14:49:01] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:02.124883
10.0.0.7 - - [19/Mar/2018 14:49:02] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:03.137896
10.0.0.7 - - [19/Mar/2018 14:49:03] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:04.152784
10.0.0.7 - - [19/Mar/2018 14:49:04] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:05.165797
10.0.0.7 - - [19/Mar/2018 14:49:05] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:06.180802
```

```
10.0.0.7 - - [19/Mar/2018 14:49:06] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:07.19569
10.0.0.7 - - [19/Mar/2018 14:49:07] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:08.210578
10.0.0.7 - - [19/Mar/2018 14:49:08] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:09.223591
10.0.0.7 - - [19/Mar/2018 14:49:09] "GET / HTTP/1.1" 200 -
2018-03-19 16:30: 10.238596
10.0.0.7 - - [19/Mar/2018 14:49:10] "GET / HTTP/1.1" 200 -
```

Log *master server*

```
2018-03-19 16:30:11.288596
10.0.0.7 - - [19/Mar/2018 14:49:11] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:12.303601
10.0.0.7 - - [19/Mar/2018 14:49:12] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:13.318489
10.0.0.7 - - [19/Mar/2018 14:49:13] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:14.331502
10.0.0.7 - - [19/Mar/2018 14:49:14] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:15.346507
10.0.0.7 - - [19/Mar/2018 14:49:15] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:16.361395
10.0.0.7 - - [19/Mar/2018 14:49:16] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:17.374408
10.0.0.7 - - [19/Mar/2018 14:49:17] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:18.389413
10.0.0.7 - - [19/Mar/2018 14:49:18] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:19.402426
10.0.0.7 - - [19/Mar/2018 14:49:19] "GET / HTTP/1.1" 200 -
```

Log *client*

```
2018-03-19 16:30:01.109878
10.0.0.7 - - [19/Mar/2018 14:49:01] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:02.124883
10.0.0.7 - - [19/Mar/2018 14:49:02] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:03.137896
```

```
10.0.0.7 - - [19/Mar/2018 14:49:03] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:04.152784
10.0.0.7 - - [19/Mar/2018 14:49:04] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:05.165797
10.0.0.7 - - [19/Mar/2018 14:49:05] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:06.180802
10.0.0.7 - - [19/Mar/2018 14:49:06] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:07.19569
10.0.0.7 - - [19/Mar/2018 14:49:07] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:08.210578
10.0.0.7 - - [19/Mar/2018 14:49:08] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:09.223591
10.0.0.7 - - [19/Mar/2018 14:49:09] "GET / HTTP/1.1" 200 -
2018-03-19 16:30: 10.238596
10.0.0.7 - - [19/Mar/2018 14:49:10] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:11.288596
10.0.0.7 - - [19/Mar/2018 14:49:11] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:12.303601
10.0.0.7 - - [19/Mar/2018 14:49:12] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:13.318489
10.0.0.7 - - [19/Mar/2018 14:49:13] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:14.331502
10.0.0.7 - - [19/Mar/2018 14:49:14] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:15.346507
10.0.0.7 - - [19/Mar/2018 14:49:15] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:16.361395
10.0.0.7 - - [19/Mar/2018 14:49:16] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:17.374408
10.0.0.7 - - [19/Mar/2018 14:49:17] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:18.389413
10.0.0.7 - - [19/Mar/2018 14:49:18] "GET / HTTP/1.1" 200 -
2018-03-19 16:30:19.402426
10.0.0.7 - - [19/Mar/2018 14:49:19] "GET / HTTP/1.1" 200 -
```

Berdasarkan tampilan log waktu diatas menunjukkan bahwa request dari *client* akan dilayani oleh *master server* ketika *master server* kembali aktif. Pada

Tabel 6.2 dibawah ini merupakan hasil pengujian *failback time* yang didapat setelah dilakukan 10 kali percobaan.

Tabel 6.2 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

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 hasil rata-rata waktu *failback time* yang didapat dari 10 kali percobaan didapatkan waktu selama 0,865 detik, seperti yang ditunjukkan.

6.2.3 Pengujian Packet Loss

Pengujian packet loss dilakukan untuk mengetahui seberapa banyak data yang hilang pada sistem saat melakukan pengiriman data. *Tool* yang digunakan adalah *Iperf*. Dengan menuliskan pada *server* perintah *iperf –s –u*. Perintah –s yang artinya digunakan pada *server* dan –u artinya menggunakan protokol UDP. dimana perintah tersebut akan dijalankan di *master server* dan *slave server*. Gambar 6.9 menunjukkan perintah tersebut telah berjalan pada *server*.

Gambar 6.9 Iperf pada server

Sama halnya seperti *server*, perintah iperf juga akan dijalankan pada *client* karena untuk pengujian menggunakan Iperf harus dipasang pada keduanya. Karena apabila dijalankan pada salah satu saja maka tidak bisa berfungsi. Dengan menulis pada *client* perintah *iperf –c 10.0.0.100 –u –t 20*. Perintah –c yang artinya digunakan pada *client*, IP 10.0.0.100 artinya IP yang akan diakses oleh *client*, –u artinya menggunakan protokol UDP, dan –t 20 yang artinya mengatur

waktu dalam 20 detik. Gambar 6.10 menunjukkan perintah tersebut telah berjalan pada *client*.

Gambar 6.10 Iperf pada Client

Pengujian *packet loss* sebelum program *failover* dilakukan dengan menggunakan tools Iperf pada *server* dan *client* agar dapat diketahui bahwa tidak terjadi *packet loss* selama program ini bekerja. Namun dalam pengujian program tersebut tidak dilakukan pemutusan jalur data pada *master server*. Setelah Iperf dijalankan pada *server* dan *client*, Hasil pengujian telah dilakukan dengan mengamati paket yang hilang. Pengujian ini melakukan percobaan selama 20 detik dan dilakukan percobaan sebanyak 10 kali. Kemudian packet loss juga dihitung pada saat *failover* bekerja, Iperf dijalankan pada *server* dan *client*, untuk mengetahui data yang hilang percobaan dilakukan dengan cara memutuskan jalur data *master server*, maka program *failover* akan bekerja. Hasil pengujian telah dilakukan dengan mengamati paket yang hilang pada saat pemutusan jalur data. Pengujian ini melakukan percobaan selama 20 detik dan dilakukan percobaan sebanyak 10 kali.

Hasil *packet loss* akan menunjukkan berapa persentase paket yang hilang selama sistem berjalan, dan hasil dari pengujian adalah sebagai berikut:

Tabel 6.3 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

Pada tabel 6.3 menunjukkan presentase paket yang hilang saat terjadi saat pengiriman data dari *client* ke *server*, dimana didapatkan rata-rata dari 10 kali percobaan dalam waktu 20 detik didapatkan hasil 0%.

Tabel 6.4 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

Pada tabel 6.4 menunjukkan persentase paket yang hilang saat terjadi saat pengiriman data dalam proses *failover*, dimana didapatkan rata-rata dari 10 kali percobaan dalam waktu 20 detik didapatkan hasil 5.6%.

BAB 7 PENUTUP

7.1 Kesimpulan

Kesimpulan pada penelitian ini adalah hasil rangkuman dari analisis yang didapatkan pada saat pengujian. Data hasil dari pengujian dan hasil pada bab sebelumnya akan ditampilkan secara ringkas dan jelas pada sub bab ini. Pada penelitian ini dapat ditarik kesimpulan sebagai berikut :

1. 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 *server* utama 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.
2. 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%.

7.2 Saran

Saran yang diperoleh dari hasil penelitian penelitian dengan judul "Implementasi *Server Failover* Pada *Software Defined Network*" adalah sebagai berikut.

1. Pada penelitian ini hanya satu sampel *client* dan pada penelitian berikutnya bisa digunakan variasi *client* yang berbeda.
2. Untuk penelitian selanjutnya bisa menambahkan beberapa *server* untuk melihat hasil packet loss.
3. Bisa dilakukan penelitian lebih lanjut terhadap *server* dengan sistem database terdistribusi dalam *server*.

DAFTAR PUSTAKA

Anees Al-Najjar, S. L. M. P., 2016. Pushing SDN to the End-Host, Network Load Balancing using *Controller*. *IEEE International Conference on Pervasive Computing and Communication Workshops.*

Durresi, M. K. &. A., 2017. Quality of Service (QoS) in Software Defined Networking (SDN): A survey. *Journal of Network and Computer Applications,* Volume 80, p. 200–218.

Fernandez, M. P., 2013. Comparing *Controller Controller* Paradigms Scalability: Reactive and proactive. *Proceedings - Advanced Information Networking and Applications (AINA),* p. 1009–1016.

Foundation, O. N., 2012. Software-Defined Networking: The New Norm for Networks [white paper]. In: *ONF White Paper.* s.l.:s.n., pp. 1-12.

Foundation, O. N., 2014. *Controller Switch* Specification. pp. 1-205.

Gibbs, M., 2005. *High availability and Heartbeat | Netwrok World.* [Online] Tersedia di: <http://www.networkworld.com/article/2321687/software/high-availability-and-heartbeat.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.

Iskandar, I., 2015. Analisa Quality of Service (QoS) Jaringan Internet Kampus. p. 10.

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.

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].

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

Nick Mckeown, G. P. T. A. L. P. H. B. J. R. J. T., 2008. *Controller*: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review,* Volume 28, p. 2.

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.

Shie-Yuan Wang, H.-W. C. C.-L. C., 2015. Comparisons of SDN *Controller Controller*s over EstiNet : Ryu vs . NOX. *The International Symposium on Advances in Software Defined Networks,* pp. 1-6.

Sulistyanto, H., 2015. Implementasi High Availability *Server* Dengan Teknik Failover Virtual Computer Cluster.

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