**Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer** e-ISSN: 2548-964X
Vol. 2, No. 8, Agustus 2018, hlm. 2297-2306 http://j-ptiik.ub.ac.id

# *High Availability Controller Software Defined Network* Menggunakan *Heartbeat* dan *DRBD*

**Maskur Purwiadi[1], Widhi Yahya[2], Achmad Basuki[3]**

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

**Abstrak**

Model arsitektur jaringan yang digunakan saat ini sulit untuk menerapkan protokol baru karena bergantung pada vendor, sehingga muncul sebuah konsep yang mengatur kinerja sistem jaringan melalui *software* bernama SDN (*Software Defined Network*). SDN berfungsi memisahkan *data-plane* dan *control-plane* pada perangkat-perangkat jaringan seperti *router* dan *switch*, sehingga pemrograman perangkat-perangkat jaringan dapat dilakukan secara terpusat melalui sebuah *controller*. *Controller* mengatur *forwarding table* (*flow-table*) *Switch* yang berfungsi meneruskan aliran paket komunikasi. Jika sebuah *controller* mati (*offline*), *switch* tidak dapat meneruskan aliran paket komunikasi dari suatu perangkat ke perangkat lain. Maka dibutuhkan suatu mekanisme untuk menjaga ketersediaan *controller yang disebut High Availabillity Controller*. *High Availability Controller* dapat dilakukan dengan mekanisme *failover* menggunakan *Heartbeat dan DRBD (Distributed Replication Block Device)*. Pengujian dalam penelitian ini dilakukan dengan metode *planned downtime*. Hasil pengujian yang diperoleh dari nilai rata – rata *downtime* POX *controller* saat terjadiproses *failover* sebesar 23 detik, dan 59,6 detik saat terjadiproses *failback*, sedangkan *Opendaylightcontroller* membutuhkan waktu sebesar 195 detik untuk proses *failover* dan 219 detik untuk proses *failback*. Metode ini memiliki kemampuan yang cukup baik, dimana semakin banyak fitur yang dijalankan sebuah *controller*, semakin lama waktu *downtime* yang dihasilkan.

**Kata Kunci***: High Availability Controller, Software Defined Network, Failover, Heartbeat, DRBD, Downtime*

**Abstract**

*Network model architecture that used nowadays are difficult to apply a new protocol because it depends upon vendor, so it appeared a concept that manage network systemperformance through software named Software Defined Network. SDN separating data-plane and control-plane in network devices such as router and switch, so that programming network devices may be done centrally through a controller.A controller manage forwarding table switch which forwarding the flow of communication package.If a controller goes offlinea switch not be able to forwarding the flow of communication package from a device to other devices. So it needs high availabillity controller mechanism to be able to maintain the availability of a controller so the communication package can be arrive. High availability controller can be done by mechanism failover use Heartbeat and DRBD.Testing in this research done with planned downtime methods. The test results that obtained from the average downtime of* POX *controller when failover process is 23 seconds, and 59,6 seconds when failback process, while Opendaylight controller took time 195 seconds for failover process and 219 seconds for faillback. This method is a good enough, where the more features run in a controller the longer time downtime will produced.*

***Keywords:*** *High Availability Controller, Software Defined Network, Failover, Heartbeat, DRBD, Downtime*

## 1. PENDAHULUAN

Teknologi jaringan internet dari tahun ke tahun berkembang sangatpesat. Berbagai macam konsep teknologi jaringan diciptakan untuk meningkatkan kinerja demi mendukung kebutuhan pengguna yang semakin kompleks. Model arsitektur jaringan yang digunakan selama ini atau yang disebut dengan *traditionalnetwork*, selalu bergantung pada *vendor* penyedia teknologi jaringan seperti CISCO, Juniper, NEC. *Vendor* tersebut

menciptakan produk-produk untuk mengatur kinerja sistem jaringan yang akan digunakan. Berawal pada tahun 2004 dicetuskan gagasan mengenai ide *New Way Managing Network* yang menginginkan suatu metode baru dalam aplikasi jaringan. Selanjutnya pada tahun 2008 gagasan awal tersebut berkembang dan melahirkan konsep yang dikenal sebagai *Software Defined Network* (SDN). Secara harfiah dapat dikatakan sebagai suatu sistem jaringan yang kinerjanya diatur oleh software tertentu. Dalam beberapa tahun perkembanganya *Software Defined Network* telah populer dan menjadi isu *global* dalam teknologi jaringan, alasanya dikarenakan SDN *platform* bersifat *universal* tanpa terbatas *vendor* atau produk tertentu dalam implementasinya. (T. Nadeau and K. Gray2013).

*Software Defined Network* (SDN) adalah sebuah konsep dalam mendesain, mengelola, dan mengimplementasikan jaringan dengan melakukan pemisahan antara *control plane* dan *forwarding plane* untuk kemudian dikendalikan melalui satu media aplikasi *controller* melewati satu protokol. *Controller Software Defined Network* bertugas mengendalikan seluruh komunikasi yang disediakan oleh sejumlah *resource* jaringan, mengendalikan *traffic* yang melewatinya serta melakukan inspeksi *traffic* atau memodifikasi *traffic* dengan mengkolaborasikan *application plane* dengan *control plane*, lalu menginstruksikan kemana suatu *traffic* atau paket data diarahkan (pada *forwading plane*). Sedangkanprotokol yang digunakan untuk komunikasi antara perangkat dan media aplikasi *controller* adalah protokol *OpenFlow*. *Openflow controller* ini seperti sebuah jembatan yang mengendalikan perangkat berdasarkan *MAC, IP Address*, atau *TCP Port* untuk melakukan perintah tertentu. (Braun Wolfgang, Menth Michael.2014).

Setiap *flowcontroller* yang diterapkan, memiliki konfigurasi tersendiri berdasarkan kebutuhan pada masing masing jaringan.Misalkan kebutuhan jaringan mengharuskan *flowcontroller* yang ditanamkan pada *switch* memiliki *idle_timeout* 10 detik, yang artinya *switch* hanya memiliki waktu 10 detik untuk tetap mengetahui arah *traffic* jaringan,setelah itu *switch* akan bertanya kembali kepada *controller*. Jika hal yang tidak diinginkan terjadi, yakni *controller* tersebut tidak aktif selama 10 menit, maka 9 menit 50 detik *host* tidak dapat terhubung. Untuk meminimalisir dampak negatif dari hal tersebut diperlukan suatu mekanisme pertahanan agar *controller* tetap *available yang* disebut dengan konsep *High Availability Controller*.

Penelitian *High Availability controller* telah dikembangkan sebelumnya oleh (John Edwart. 2017). Yaitu dengan menggunakan *Fault Tolerance Multi Controller*. *Fault Tolerance Multi Controller* merupakan sebuah metode yang memungkinkan suatu sistem tetap berjalan normal yang memanfaatkan kelebihan *Floodlight Multi Controller*, dengan membagi tugas *controller* menjadi dua yaitu sebagai *primary* dan *backup*. Dalam penelitian tersebut dihasilkan nilai rata-rata waktu yang diperlukan untuk mekanisme *failover*, dimana semakin banyak *controller* yang digunakan maka semakin lama waktu yang diperlukan untuk melakukan sistem *failover controller*.

*Fault Tolerance Multi Controller* menempatkan *controller primary* dan *backup* pada satu *server*, jika *server* tersebut mati dengan asumsi penyebab *server* mati adalah padam nya listrik pada lingkungan tersebut serta tidak adanya cadangan listrik yang dapat digunakan, maka diperlukan *server* lain yang siap untuk mem-*backup* kinerja *controller* tersebut. Dalam hal ini telah dikembangankan suatu mekanisme untuk meningkatkan *High Availability* yang diimplementasikan pada *server* menggunakan *Heartbeat dan DRBD (Distributed Replicated Block Device)* oleh (Miers Karin, 2004). Dalam penelitian tersebut*Heartbeat* melakukan deteksi ketika terjadi kesalahan atau *down* pada *server* serta memindahkan *server primary* ke *backup*. Sedangkan DRBD (*Distributed Replicated Block Device)* melakukan *backup* keseluruhan dari *server primary* ke *server backup* secara *realtime*. Data *delay* dan *troughput* yang dihasilkan dalam penelitian menunjukkan metode tersebut berjalan dengan baik dengan waktu *take over* yang singkat. Konsep *High Availability* menggunakan *Heartbeat* dan *DRBD (Distributed Replicated Block Device)* memungkinkan tetap dilakukannya mekanisme *Failover* dengan lokasi *primary* dan *backup* yang berjauhan, untuk menghindari gangguan dalam satu lingkungan.

Berdasarkan hal yang sudah dijelaskan diatas, penulis melakukan penelitian ini. Parameter yang akan diujikan dalam penelitian ini adalah waktu rata rata *failovercontroller* dimana *controller* utama mati dan tugas diambil alih oleh *controllerbackup,dan waktu rata-rata failback controller* dimana *controller* utama

kembali aktif dan tugas dikembalikan kepada *controller* utama.

## 2. LANDASAN KEPUSTAKAAN

Penelitian-peneltian yang dilakukan sebelumnya menerapkan sistem *High Availability Controller* dengan menggunakan berbagai mekanisme. Pada penelitian ini *menggunakan High Availability Controller* dilakukan dengan menerapkan mekasme *failover* menggunakan *Heartbeat* dan DRBD.

### 2.1 *High Availability*

*High Availability* merupakan kemampuan satu sistem atau kelompok sistem (*cluster*) menjaga aplikasi (*application*) atau layanan (*service*) berjalan. Perancangan untuk ketersediaan mengasumsikan sistem-sistem akan gagal dan sistem-sistem dikonfigurasi untuk menutupi (mask) dan dapat memulihkan diri dari kegagalan komponen atau sistem dengan dampak penghentian aplikasi yang minimum. Pada konsep *High Availability* terdapat dua jenis, yakni *Continous Availability* dan *Failover Availability*. (Andreas Weininger. 2013)

*Continuous Availability* merupakan sebuah konsep dimana *service* (sebagai contoh *database engine*) dituntut untuk tetap *available* tanpa adanya *downtime* yang dapat dirasakan oleh *user*. Terdapat tiga desain standar yang dapat digunakan untuk mengimplementasikan *Continous Availability*, yaitu *Programatic*, *Shared* dan *Multiple Copy*. *Programatic* merupakan desain dengan memanfaatkan pemrograman dari sisi *user application* dengan cara mengakses ke dua lebih *server* yang tidak saling berhubungan. Keuntungan dari sistem ini adalah mendapatkan *maximum flexibility* dimana saat terjadi *failure* pada satu *server* tidak akan mempengaruhi kinerja dari *user application. Shared Availability* merupakan desain dengan penggunaan *redundant* sistem yang berbagi *criticalresources*. Sedangkan *Multiple Copy* adalah desain yang menggunakan beberapa *copy* data untuk menjaga ketersediaan beberapa *server* yang saling berhubungan. (Andreas Weininger. 2013)

*Failover Availability* merupakan sebuah konsep yang berbeda dari *Continous Availability*, dimana *Failover Availability* membutuhkan waktu saat terjadi*failure* meskipun dalam kurun waktu yang singkat. Dalam konsep ini digunakan dua buah *server* yakni *server* utama dan *server backup* dengan data yang identik pada masing-masing *server*. Pada saat sistem dengan konsep ini berjalan normal, hanya *server* utama yang bertugas untuk melayani seluruh *user*, sedangkan pada saat *server* utama mengalami *failure* dan *server backup* mendeteksi hal tersebut, maka *server backup* akan menggantikan fungsi dari *server* utama. *Failover Availability* mempunyai dua tipe yakni *Synchronous* dan *Asynchronous*. Pada tipe *Synchronous* masing-masing *server* (*server* utama dan *server backup*) saling melakukan sinkronisasi data sehingga keseragaman data pada masing-masing *server* terjamin. Sedangkan tipe *Asynchronous* sinkronisasi data dilakukan dengan cara mengirimkan perubahan data dari *server* yang sedang aktif menuju *server* yang sedang pasif sehingga hanya satu *server* yang melakukan sinkronisasi. (Andreas Weininger. 2013)

### 2.2 *Software Defined Network*

*Software Defined Network* (SDN) adalah sebuah paradigma baru dalam mengontrol dan manajemen jaringan komputer. Konsep dasar dari SDN adalah pemisahan antara *controlplane* yang bertugas untuk menentukan bagaimana sebuah paket akan diteruskan dengan *dataplane* yang bertugas meneruskan paket (Al-Najjar et al. 2016). Tekno*log*i *Software Defined Network* (SDN) memisahkan antara *Data Plane* dan *Control Plane*. Dengan *control plane* yang terpisah, *administrator* dapat memodifikasi protokol sepanjang perangkat jaringan. *Controller* saat ini yang paling banyak digunakan adalah POX. Data plane merepresentasikan perangkat penerus *traffic* dalam arsitektur *Software Defined Network* (SDN). Karena *controller* memerlukan komunikasi dengan infrastruktur jaringan, diperlukan protokol untuk mengendalikan dan mengatur antarmuka antar berbagai macam perangkat jaringan. Protokol yang paling banyak digunakan adalah *OpenFlow*. (Nadeau & Gray 2013).

*Controller* mengambil alih keputusan-keputusan yang biasa dilakukan oleh *control plane*, sepert *routing, switching, Quality of Service (QoS)*, dan lain- lain. Otomasisasi dimungkinkan dengan dikembangkannya API yang mendeteksi perubahan dalam *flow* atau jaringan. Misal, jika sebuah *VM* dipindahkan dari satu *switch* ke *switch* yang lain, maka *routing atau Quality of Service (QoS)* akan ikut

berubah secara otomatis. SDN memungkinkan *network administrators* untuk memprogram pusat kontrol jaringan melalui sebuah *controller* tanpa akses fisik ke *switch*. (T. Nadeau and K. Gray. 2013)

### 2.3 *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 *engressport* pada suatu *device*. *Dataset* yang digunakan untuk menyimpan topologi jaringan disebut *RoutingInformationBase* (RIB). Setelah RIB terbentuk dan stabil, RIB tersebut kemudian akan digunakan untuk memprogram *ForwardingInformationBase* (FIB) yang digunakan oleh *data plane* untuk meneruskan paket (Nadeau & Gray 2013). Dalam implementasinya *controller* dapat bekerja dengan tiga mode, *reactive mode*, *proactive mode*, dan *hybrid mode*(Fernandez 2013). Mode tersebut bekerja sebagai berikut :

### 2.4 *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*(Nadeau & Gray 2013). *Data plane* merupakan lapisan terendah dari SDN yang berisi *network device* seperti *router, physical/virtual switch, access point* dll. *Data plane* dapat berkomunikasi dengan SDN *controller* menggunakan protokol *OpenFLow*(Karakus & Durresi 2017).

### 2.5 *OpenFlow Protokol*

*OpenFlow* adalah *interface* komunikasi standar pertama 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*(Open Network Foundation 2012). Terdapat tiga bagian pada *switch* didalam arsitektur SDN dengan protokol *OpenFlow*. Bagian tersebut adalah *Flow Table, Secure Channel,* danOpenflow *Protokol* (Karakus & Durresi 2017). *OpenFlowchannel* merupakan sebuah *interface* yang menghubungkan komunikasi antara *OpenFlowcontroller* dengan *OpenFlowswitch*. 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 protokol*. (Open Network Foundation 2014)

### 2.6 Mekanisme *Failover*

Mekanisme *failover* adalah suatu teknik jaringan dengan memberikan dua jalur koneksi atau lebih dimana ketika salah satu jalur mati, maka koneksi masih tetap berjalan dengan mengalihkan ke jalur lainnya. Pada proses peralihan, dialihkan sebuah komponen cadangan, elemen, atau operasi, sementara perbaikan untuk mengatasi gangguan sedang dijalankan. Prosedur mekanisme *failover* menentukan kelangsungan operasional jaringan. Mekanisme *failover* dapat dirancang sehingga dapat sesegera mungkin bertindak setelah terjadigangguan muncul. (Purnomo Nanang. 2012).

### 2.7 *Distributed Replicated Block Device*

*DRBD (Distributed Replicated Block Device)* adalah *storage block device* yang dirancang untuk membangun *High Availability sistem*. Kinerja DRBD *(Distributed Replicated Block Device)* dilakukan dengan cara *mirroring* terhadap semua melalui jaringan serta menggunakan jaringan sebagai media transmisi untuk melakukan *sinkronisasi* data antar *disk* secara *realtime*. *DRBD* bisa dianalogikan sebagai mekanisme *RAID-1* (mirroring, bisa juga tipe *RAID* lain yang menggunakan prinsip *mirroring*), yang melakukan duplikasi data melalui *network*. Duplikasi data ini dilakukan dalam mekanisme *blockdevices*, bukan dalam bentuk data mentah. Jika *RAID-1* melakukan duplikasi isi dan data suatu *harddisk* atau partisi ke *harddisk* atau partisi lain, *DRBD* melakukan hal yang sama, hanya saja dilakukan melalui *network*. *DRBD* dan *harddiskRAID* bersifat saling mendukung. *DRBD* memiliki satu keunggulan dibandingkan *harddiskRAID*, yaitu *backupserver* berada terpisah dengan sumber *backup*. Pemisahan ini membawa keuntungan preventif, jika ada masalah pada salah satu

*server*, *server* lainnya akan bertindak sebagai *server* pengganti. Jika *server* utama sudah kembali pulih, kendali akan dikembalikan ke *server* utama. (Haas, Florian. 2011)

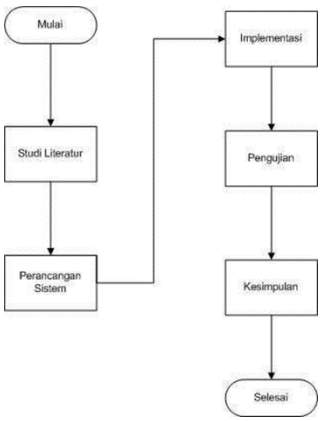
### 2.8 *Heartbeat*

*Heartbeat* merupakan sebuah aplikasi yang dapat mendeteksi apabila *server* utama *down* maka *Heartbeat* akan secara otomatis mengarahkan peran *server* utama kepada *server backup*. *Heartbeat* menjalankan script inisialisasi untuk menjalankan *service* lain saat *Heartbeat* dijalankan atau bisa juga mematikan *service* lain saat *Heartbeat* dimatikan. (M. Tim Jones. 2012)

*Heartbeat* melakukan pemeriksaan suatu jalur atau koneksi bila terjadikerusakan pada salah satu link atau koneksi dan dikirimkan ke antara *node* untuk menjamin ketersedian setiap *node*. (Marowsky-Bree, L. (2004) 34).

## 3. METODOLOGI PENELITIAN

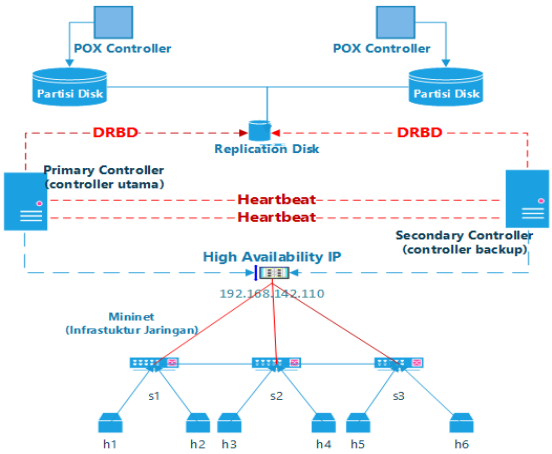
Diagram alir dari keseluruhan proses penelitian dapat dilihat pada Gambar 1.




**Gambar 1** Diagram Alir Metodologi Penelitian

## 4. PERANCANGAN

Perancangan *sistemHigh Availability Controller* SDN (*Software Defined Network*) ini dibagi menjadidua tahap, perancangan *hardware* dan perancangan *software*. Analisis kebutuhan dilakukan untuk mengetahui segala kebutuhan yang akan digunakan untuk perancangan *sistemHigh Availability Controller* SDN. Untuk perancangan *Hardware* terdiri atas perancangan *Servercontroller* SDN, perancangan *server* Mininet, spesifikasi *servercontroller*, spesifikasi *server* Mininet, dan fungsi dari seluruh *server* yang dijalankan. Sedangkan untuk perancangan *software* terdiri atas instalasi sistem operasi Ubuntu, instalasi DRBD (*Distributed Replication Block Device*), instalasi POX *Controller*, instalasi *Heartbeat*, instalasi Mininet. Berikut gambar rancangan keseluruhan sistem *High Availability Controller Software Defined Network*.




**Gambar 2** Arsitektur High Availability *Controller* Software Defined *Network*

Berdasarkan gambar 2, akan dibuat 3 *VM(Virtual Machine*), yakni *VM* untuk *controller* utama, *VM* untuk *controllerbackup*, dan *VM* untuk Mininet. Pada *VMcontroller* utama dan *controller backup*, disediakan *partisi* yang identik. *Controller* ditempatkan pada *partisi* yang telah dibuat, untuk selanjutnya DRBD akan melakukan sinkronisasi data pada partisi *disk* yang telah dibuat secara *realtime*, membentuk *replicationdisk*. Replication *disk*, akan diakses secara bergantian oleh *server* menggunakan *Heartbeat*, dimana jika *server* utama yang aktif, maka *replicationdisk* akan diakses oleh *server* utama. Jika *server* utama mati, maka *replicationdisk* diambil alih hak akses nya oleh *server backup*. *Heartbeat* juga bertugas membuat IP baru dimana, jika *controller* utama (*server* utama) mati, Mininet sebagai infrastruktur jaringan, akan mengarah ke IP yang telah dibuat oleh *Heartbeat* pada *controller backup*. Sehingga terbentuk suatu *High AvailabilityIP* yang nantinya akan menjadiakses *remote controller* pada Mininet untuk terhubung dengan POX *Controller*.

## 5. IMPLEMENTASI

### 5.1 Implementasi *VMware*

VMware yang digunakan dalam penelitian ini adalah VMware® Workstation 12 Pro. Dalam implementasi ini dibutuhkan tiga buah *Virtual Machine*. *Virtual Machine* pertama bertugas sebagai *primaryserver* atau *controller*

utama, *Virtual Machine* kedua bertugas sebagai *secondaryserver* atau *backup* dari *controller* utama. Sedangkan *Virtual Machine* ketiga bertugas sebagai infrastruktur topologi yang akan dijalankan dengan Mininet.

### 5.2 Implementasi *Distributed Replication Block Device*

Proses instalasi (*Distributed Replication Block Device*) DRBD ini dilakukan dengan mengunduh langsung aplikasi DRBD dari *repository* dengan melakukan perintah apt-get install drbd8-utils pada terminal. Setelah proses selesai DRBD siap untuk dikonfigurasi.

Proses konfigurasi DRBD (*Distributed Replication Block Device*), terdapat pada file /etc/drbd.conf yang dapat diedit dengan perintah nano drbd.conf.

### 5.3 Implementasi POX*Controller*

POX merupakan *controller* yang berbasis python. Sebelum proses instalasi POX, pastikan *python* sudah terinstall terlebih dahulu. POX *controller* dapat diinstall dengan cara *clone* dari *repository*. POX *controller* telah selesai diinstall, selanjutnya buka direktori POX dengan perintah #cd POX.

Langkah selanjutnya dilakukan perintah seperti berikut:

```
./POX.py forwarding.l2_learning
openflow.of_01 --port=6633 --
address=192.168.142.110
```

Perintah diatas dilakukan untuk menjalankan file POX.py dengan konfigurasi *controller*, IP Adress dan *port* yang ditentukan. Pada penelitian ini digunakan *port* 6633 dan IP Address 192.168.142.110 yakni *IP Address* dari *primaryserver* yang nantinya akan digunakan oleh topologi jaringan.

### 5.4 Implementasi *Heartbeat*

Implementasi *Heartbeat* dilakukan dengan cara konfigurasi *Heartbeat* pada file authkeys, ha.cf, dan haresources. Pada proses konfigurasi *Heartbeat* ini harus dibuat tiga file utama. Tiga file tersebut adalah authkeys, ha.cf dan haresources yang diletakkan pada folder /etc/ha.d. File ha.cf berfungsi sebagai konfigurasi dasar *Heartbeat*. File haresources berfungsi sebagai sumber (*resources*) dari *Heartbeat*, berisikan node, ip address, drbd disk dan *service* yang akan dijalankan. File authkeys berfungsi sebagai kunci autentifikasi dari *Heartbeat*, berisikan *password* yang digunakan untuk akses komunikasi antar *server*.

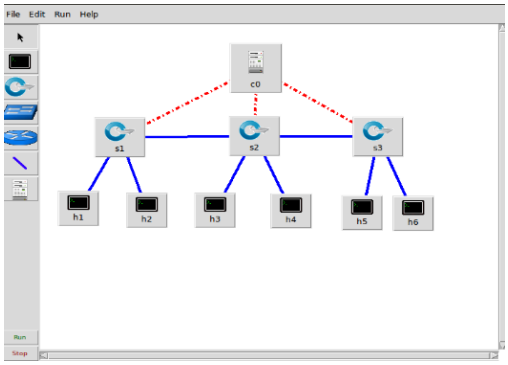
Pada proses menjalankan *Heartbeat* diperlukan beberapa *Init-scripts* yang digunakan untuk mengatur *service* yang akan dijalankan. *Init-scripts* tersebut disimpan pada /etc/ha.d/*resource*.d/. Konfigurasi file tersebut dilakukan pada *primaryserver* dan *secondaryserver*.

### 5.5 Implementasi *Mininet*

Mininet mempunyai fitur miniedit yang berbasis GUI untuk memudahkan user dalam mengatur konfigurasi infrastruktur yang akan dibuat. Penelitian ini dilakukan dengan memanfaatkan fitur tersebut. Berikut perintah untuk menjalankan *miniedit*:

```
root@Mininet:/home/maskur# sudo
python ./Mininet/examples/miniedit.py
```

Memanfaatkan miniedit dapat mempermudah pembuatan infrastruktur jaringan, serta dapat menyimpan infrastruktur yang telah dibuat. Pada penelitian ini telah dibuat infrastruktur sederhana dan disimpan pada direktori Mininet. Berikut infrastruktur yang telah dibuat:




**Gambar 3** Infrastruktur jaringan pada Mininet.

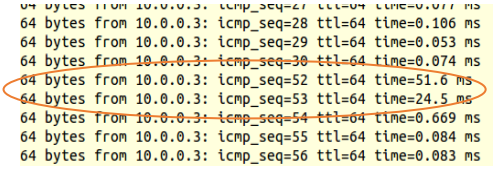
Berdasarkan gambar 3, Topologi yang dibuat adalah topologi sederhana yang memiliki satu *controller*, tiga *switch* dan enam *host*. *Controller* yang dibuat akan melakukan *remote* ke POX *controller* yang telah dijalankan. Langkah berikutnya adalah konfigurasi *controller*. Sebelum konfigurasi *controller*, pastikan POX *controller* pada *server* utama (*primaryserver*) sudah berjalan dengan benar. Pada konfigurasi tersebut *controller* akan me-*remoteport* pada IP POX *controller* yang telah dikonfigurasi sebelumnya. *Port* yang digunakan adalah 6633, sedangkan *IP Address* yang digunakan adalah 192.168.142.110.

## 6. PENGUJIAN DAN ANALISIS

### 6.1 Pengujian Mekanisme *Failover*

Pengujian *Failover* ini dilakukan dengan cara mematikan *server* utama secara langsung (power off *Virtual Machine*), dengan kondisi seperti ini maka semua *resource*s dari *server* utama akan segera dipindahkan ke *server* kedua dan menjalankan *service* pada *server* kedua untuk melakukan tugasnya seperti *server* utama.

Berikut hasil yang diperoleh dari *ping* h1 – ke h3 selama proses *failover* berlangsung.




**Gambar 4** Interkoneksi *host* (h1 *ping* h3) pada saat proses *failover*

Berdasarkan gambar 4, ketika *icmp_seq*=30 *switch* tidak bisa terhubung dengan *controller*, sehingga proses *ping* berhenti. Selama proses ini *switch* akan mencoba terhubung dengan *controller*. Setelah beberapa detik *ping* dapat dilanjutkan dengan *icmp*_seq=52. Setelah *Heartbeat* DRBD untuk melakukan takeover *resource* dari *replication disk* yang berada pada *primaryserver*. Muncul tampilan direktori *replication disk*, yang menandakan bahwa *takeover resource drbddisk* telah selesai dilakukan. Selanjutnya *Heartbeat* akan memberikan perintah pada POX *controller* untuk menjalankan fungsinya. Waktu yang dibutuhkan untuk keseluruhan proses *failover* POX *controller*, dapat dilihat berdasarkan waktu yang dibutuhkan *switch* untuk terhubung kembali dengan *controller*. Hal tersebut dapat dilihat pada *log open switch* yang terdapat pada file /var/*log*/ open*switch*/ovs-vswicthd.*log*.

**Tabel 1** Hasil *Log open switch* ketika menggunakan POX *Controller*

| *LogSwitch* | Waktu |
|---|---|
| \|00229\|rconn\|ERR\|s1<->tcp:192.168.142.110:6633: no response to inactivity probe after 5 seconds, disconnecting | 2017-07-05 10:20:48 |
| \|00230\|rconn\|ERR\|s2<->tcp:192.168.142.110:6633: no response to inactivity probe after 5 seconds, disconnecting | 2017-07-05 10:20:49 |
| \|00231\|rconn\|ERR\|s3<->tcp:192.168.142.110:6633: no response to inactivity probe after 5 seconds, disconnecting | 2017-07-05 10:20:50 |
| \|00232\|rconn\|INFO\|s1<->tcp:192.168.142.110:6633: connected | 2017-07-05 10:21:15 |
| \|00233\|rconn\|INFO\|s2<->tcp:192.168.142.110:6633: connected | 2017-07-05 10:21:15 |
| \|00234\|rconn\|INFO\|s3<->tcp:192.168.142.110:6633: connected | 2017-07-05 10:21:15 |

Berdasarkan *log* diatas, pada pengujian pertama dibutuhkan waktu 25 detik oleh *switch* dapat terhubung kembali dengan POX *controller*. Pada penelitian ini dilakukan 10 kali pengujian, dan didapatkan hasil sebagai berikut:

**Tabel 2** Hasil Keseluruhan Pengujian POX *controller* pada proses *failover*.

| Pengujian | Waktu yang dibutuhkan |
|---|---|
| 1 | 25 detik |
| 2 | 30 detik |
| 3 | 32 detik |
| 4 | 27 detik |
| 5 | 25 detik |
| 6 | 27 detik |
| 7 | 27 detik |
| 8 | 25 detik |
| 9 | 30 detik |
| 10 | 25 detik |
| Rata – rata | 27.3 detik |

Hasil pengujian menunjukkan waktu yang cukup singkat untuk POX *controller* dapat aktif kembali setelah terjadi kegagalan pada *primary server*. Waktu yang dibutuhkan sebuah *controller* untuk dapat aktif kembali juga bergantung pada berapa banyak fitur yang harus dijalankan oleh *controller* tersebut. Dalam hal ini telah diujikan sebelumnya dengan konfigurasi mekanisme *failover* yang sama untuk *Opendaylight Controller*, dimana pada pengujian pertama dibutuhkan waktu 3 menit 12 detik untuk proses *failover Opendaylight Controller*. Berikut hasil *log switch* dan keseluruhan pengujian *failover* menggunakan *Opendaylight Controller*:

**Tabel 3** Hasil Keseluruhan Pengujian *Opendaylight Controller* pada proses *failover*

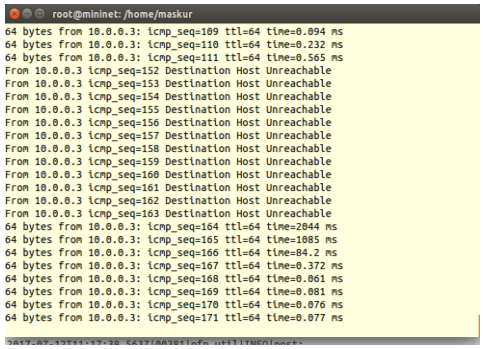
| Pengujian | Waktu |
|---|---|
| 1 | 3 menit 12 detik |
| 2 | 3 menit 17 detik |
| 3 | 3 menit 14 detik |
| 4 | 3 menit 12 detik |
| 5 | 3 menit 15 detik |
| 6 | 3 menit 15 detik |
| 7 | 3 menit 19 detik |
| 8 | 3 menit 20 detik |
| 9 | 3 menit 20 detik |
| 10 | 3 menit 14 detik |
| Rata – rata | 3 menit 15.8 detik |

Setelah pengujian *failover* berhasil, selanjutnya dilakukan pengujian *failback*. Pengujian *failback* adalah pengujian yang dilakukan ketika *primary server* telah aktif kembali untuk selanjutnya *resource* dari *secondary server* diambil alih kembali oleh *primary server*. Pengujian *failback* dapat dilakukan dengan melakukan perintah penghentian *Heartbeat* pada *server* kedua dengan cara mengetikkan perintah "*service Heartbeat* stop" pada terminal *server* 2.

Proses *failback* hampir sama dengan proses *failover*, yang membedakan hanya pengembalian *resource* dari *secondary server* menuju *primary server*. Proses *failback* membutuhkan waktu yang relatif lebih lama daripada proses *failover*, karena dalam proses *failback* membutuhkan pengecekan apakah data pada *replication disk* sudah sinkron (*Up to Date*). Setelah proses pengecekan tersebut selesai maka akan dipindahkan *service* yang dijalankan pada *server* kedua kembali menuju *server* utama.

Ketika proses perpindahan dan pengecekan *resource*, maka *controller* belum dapat diakses oleh jaringan. Dibutuhkan waktu untuk menjalankan kembali POX *controller* pada *primaryserver* atau *server* utama.

Setelah *resource* dari *secondaryserver* telah selesai dikembalikan ke *primaryserver*, proses dilanjutkan dengan menjalankan POX *controller*. Ketika POX *controller* telah berhasil dijalankan, *ping* h1 ke h3 yang dilakukan oleh Mininet mulai berjalan kembali. Berikut hasil tampilan bahwa POX *controller* sudah dapat diakses kembali oleh *switch*.




**Gambar 5** Proses *ping* h1 – h3 saat terjadi mekanisme *failback*

Berdasarkan proses *ping* pada gambar 5, pada saat icmp_seq=111, *primaryserver* menyatakan bahwa sudah kembali aktif, dan *resource* sudah dapat dikembalikan. Sampai pada icmp_seq=164 *ping* sudah dapat dilanjutkan, dengan begitu proses pengembalian *resource* telah berhasil dan POX *controller* sudah berhasil dijalankan kembali. Untuk mengukur waktu yang dibutuhkan dalam proses *failback* ini, dapat dilihat pada *log* open*switch*.

Berdasarkan *log* openswitch, pada pengujian pertama dibutuhkan waktu 57 detik oleh *switch* dapat terhubung kembali dengan POX *controller*. Pada penelitian ini dilakukan 10 kali pengujian, dan didapatkan hasil sebagai berikut:

**Tabel 4** Hasil Keseluruhan Pengujian POX *Controller* pada proses *Failback*

| Pengujian | Waktu yang dibutuhkan |
|---|---|
| 1 | 57 detik |
| 2 | 61 detik |
| 3 | 59 detik |
| 4 | 60 detik |
| 5 | 62 detik |
| 6 | 60 detik |
| 7 | 60 detik |
| 8 | 57 detik |
| 9 | 59 detik |
| 10 | 61 detik |
| Rata – rata | 59.6 detik |

Seperti halnya proses *Failover*, untuk proses *Failback* juga dibutuhkan waktu tambahan untuk menjalankan keseluruhan fitur dari POX *Controller*. Namun pada proses *failback* membutuhkan waktu yang sedikit lebih lama. Hal yang sama berlaku jika menggunakan *controller* lain seperti *Opendaylight*, dimana masing-masing *controller* memiliki fitur yang

berbeda, sehingga mempengaruhi waktu digunakan. Berikut hasil keseluruhan pengujian menggunakan *OpendaylightController*.

**Tabel 5** Hasil Keseluruhan Pengujian *OpendaylightController* pada proses *Failback*

| Pengujian | Waktu yang dibutuhkan |
|---|---|
| 1 | 3 menit 37 detik |
| 2 | 3 menit 40 detik |
| 3 | 3 menit 41 detik |
| 4 | 3 menit 37 detik |
| 5 | 3 menit 39 detik |
| 6 | 3 menit 41 detik |
| 7 | 3 menit 41 detik |
| 8 | 3 menit 42 detik |
| 9 | 3 menit 41 detik |
| 10 | 3 menit 40 detik |
| Rata-rata | 3 menit 39.9 detik |

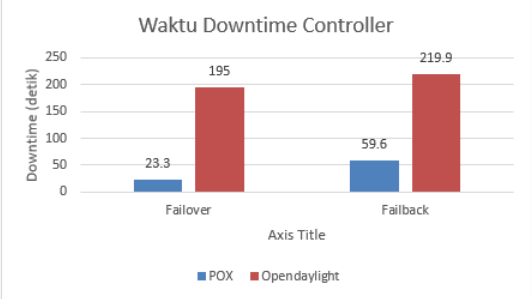
## 6.2 Analisis Sistem *High Availability Controller*

Pengujian ini dilakukan untuk membuat *High Availability Controller Software Defined Network*, dimana setiap percobaan dilakukan untuk mendapatkan *downtime controller* ketika *server controller* mengalami *down* atau mati.

Berdasarkan hasil keseluruhan waktu yang didapat, hanya dibutuhkan waktu kurang dari 30 detik untuk proses *failover* dan kurang dari 65 detik untuk proses *failback*. Konsep *High Availability Controller Software Defined Network* menggunakan *Heartbeat* dan DRBD cukup mumpuni untuk menjaga ketersediaan *controller* yang dibutuhkan. Serta banyak-nya *controller* dalam infrastruktur topologi tidak akan berpengaruh terhadap waktu yang dibutuhkan untuk proses *failover* dan *failback*. Dikarenakan *Controller* telah direplikasi oleh *server backup* secara *realtime*, dan *resource* dipindahkan dari *server* yang mati ke *server* yang aktif.

Adapun faktor yang mempengaruhi waktu yang dibutuhkan untuk proses *failover controller Software Defined Network* ini adalah:

1. Fitur yang dijalan oleh masing – masing *controller*, dimana semakin banyak fitur yang di-generate, semakin bertambah waktu *downtime* yang dihasilkan. Berikut grafik waktu *downtime* yang dihasilkan oleh POX dan *Opendaylight Controller*.




**Gambar 6** Grafik rata-rata Downtime *Controller* SDN.

Salah satu contoh yang membuat perbandingan tersebut adalah, adanya fitur GUI bernama DLUX yang terdapat pada *Opendaylight controller*, sedangkan POX *controller* hanya berupa *console*.

2. Besarnya *resource* pada *replication disk* yang harus dipindahkan ketika terjadi *failover* dan *failback*. Hal tersebut terjadiketika terdapat banyak data yang terdapat pada *replication disk*, dimana ketika terjadi *failover* atau *failback* data tersebut mengalami banyak perubahan, sehingga diperlukan waktu tambahan untuk mencocokkan setiap data yang terdapat pada *replication disk*.
3. Banyaknya *service* yang harus dijalankan oleh *Heartbeat*, dimana *Heartbeat* akan menjalankan *service* yang sudah diinisialisasikan pada konfigurasi file *haresources* secara bertahap sesuai urutan yang telah dibuat.

## 7. KESIMPULAN

Berdasarkan perancangan, impementasi, dan pengujian yang dilakukan pada *server* utama, *server* kedua, dan *server* Mininet. Maka disimpulkan bahwa:

1. Mekanisme *failover controller* menggunakan *Heartbeat* dan DRBD (*Distributed Replication Block Device*) dilakukan dengan membagi peran *controller* menjadi *controller* utama dan *controller backup*. Jika *controller* utama mati, maka *controller backup* akan mengambil alih tugas *controller* utama.
2. Kecepatan waktu dalam proses failover dan failback memerlukan rata-rata waktu downtime yang relatif singkat, yaitu 27,3 detik untuk proses failover, dan 59,6 detik untuk proses failback pada POX *controller*. Sedangkan pada saat menggunakan *Opendaylight controller* rata-rata waktu

downtime yang dihasilkan yaitu 3 menit 39,9 detik untuk proses failover dan 219,9 detik. Perbedan hasil tersebut bergantung pada jenis *controller* yang digunakan, dimana semakin banyak fitur yang dijalankan sebuah *controller*, semakin lama waktu downtime yang dihasilkan. Hal tersebut terjadikarena dibutuhkan waktu tambahan untuk menjalankan fitur-fitur *controller* yang digunakan.

## DAFTAR PUSTAKA


Al-Najjar, A., Layeghy, S. &*Port*mann, M., 2016. Pushing SDN to the end-host, network load balancing using OpenFlow. *2016 IEEE International Conference on Pervasive Computing and Communication Workshops, PerCom Workshops 2016*.

Braun Wolfgang, Menth Michael. 2014, "Software-Defined *Network*ing Using OpenFlow: Protocols Applications and Architectural Design Choices". Future Internet

Bourke Tony. 2001. "*Server* Load Balancing, O'Reilly&Associates", Inc.

Fernandez, M.P., 2013. Comparing OpenFlow *controller* paradigms scalability: Reactive and proactive. *Proceedings-International Conference on Advanced Information Networking and Applications, AINA*, pp.1009–1016.

Haas,Florian. 2010. "*The HeartbeatUser Guide*." TheLinux-HA*User*'sGuide.

Haas,Florian.2011."*TheDRBDUser'sGuideversion 1.3.4*". LINBITInformation Techno*log*iesGmbH.

John Edwart. 2017. "Analisis *Failover* Multi *Controller* Berbasis Floodlight *Controller* pada *Software Defined Network* (SDN)". Fakultas Informatika dan Ilmu Komputer Universitas Brawijaya: Malang.

Karakus, M. & Durresi, A., 2017. Quality of Service (QoS) in Software Defined Networking (SDN): A survey. *Journal of Network and Computer Applications*, 80(November 2016), pp.200–218. Available at: http://dx.doi.org/10.1016/j.jnca.2016.12.019.

Marowsky-Bree,L.(2004),"A New Cluster *Resource* Manager for *Heartbeat*", *Proceedings of UKUUG LISA/Winter Conference High-Availability and Reliability*, The UK's Unix & Open Systems *User* Group, Bournemouth, UK.

M. Tim Jones. 2012."High-availability storage with Linux and DRBD" Open *Network*ing Foundation. "*Software*-Defined *Network*ing: The New Norm for *Network*s," ONF White Paper.

Open Network Foundation, 2012. Software-Defined Networking: The New Norm for Networks [white paper]. *ONF White Paper*, pp.1–12.

Purnomo Nanang. 2012. "Pemanfaatan *Failover* Cluster *Server* Guna Recovery Sistem Pada PT. Lintas Data Prima", Sekolah Tinggi Manajemen Informatika dan Komputer AMIKOM: YOGYAKARTA.

T. Nadeau and K. Gray, "SDN," O'Reilly, 2013, 384 pp, ISBN:978-1-449-34230-2 (Safari book).

Wang, S., 2015. Comparisons of SDN OpenFlow *Controller*s over EstiNet : Ryu vs . NOX. *The International Symposium on Advances in Software Defined Networks, April 19-24, 2015, Barcelona, Spain*, (Fedora 14), pp.1–6.