

Implementasi Dynamic Switch Migration pada Controller Terdistribusi di Software Defined Network.

Tugas Akhir

diajukan untuk memenuhi salah satu syarat

memperoleh gelar sarjana

dari Program Studi Teknik Informatika

Fakultas Informatika

Universitas Telkom

1301144006

Rizal Mochamad Nazar



Program Studi Sarjana Teknik Informatika

Fakultas Informatika

Universitas Telkom

Bandung

2018

LEMBAR PENGESAHAN

Implementasi Dynamic Switch Migration pada Controller Terdistribusi di Software Defined Network.

Implementation Dynamic Switch Migration on Distributed Controller in Software Defined Network.

NIM: 1301144006

Rizal Mochamad Nazar

Tugas akhir ini telah diterima dan disahkan untuk memenuhi sebagian syarat memperoleh gelar pada Program Studi Sarjana Teknik Informatika

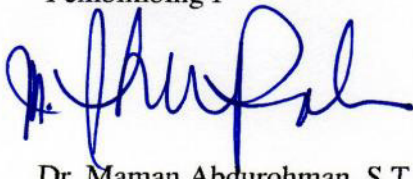
Fakultas Informatika

Universitas Telkom

Bandung, 9 Agustus 2018

Menyetujui

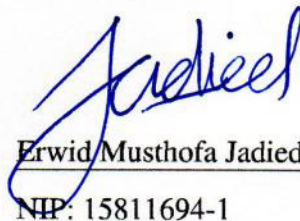
Pembimbing I



Dr. Maman Abdurohman, S.T., M.T

NIP: 99750180-1

Pembimbing II



Erwid Musthofa Jadied, S.T., M.T

NIP: 15811694-1

Ketua Program Studi

Sarjana Teknik Informatika,
a.n Bambang Ari W



Said Al Faraby, S.T., M.Sc

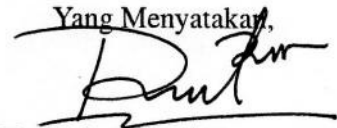
NIP: 15890019

LEMBAR PERNYATAAN

Dengan ini saya, Rizal Mochamad Nazar, menyatakan sesungguhnya bahwa Tugas Akhir saya dengan judul ” **Implementasi Dynamic Switch Migration pada Controller Terdistribusi di Software Defined Network.** ” beserta dengan seluruh isinya adalah merupakan hasil karya sendiri, dan saya tidak melakukan penjiplakan yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan. Saya siap menanggung resiko/sanksi yang diberikan jika dikemudian hari ditemukan pelanggaran terhadap etika keilmuan dalam buku TA atau jika ada klaim dari pihak lain terhadap keaslian karya.

Bandung, 9 Agustus 2018

Yang Menyatakan,



Rizal Mochamad Nazar

Implementasi Dynamic Switch Migration pada Controller Terdistribusi di Software Defined Network.

Rizal Mochamad Nazar¹, Maman Abdurohman², Erwid Musthofa Jaded³

^{1,2,3}Fakultas Informatika, Universitas Telkom, Bandung

¹rizal@students.telkomuniversity.ac.id, ²abdurohman@telkomuniversity.ac.id,

³jaded@telkomuniversity.ac.id,

Abstrak

Software Defined Network merupakan teknologi yang dapat mengelola jaringan skala besar dengan memisahkan *control plane* dan *data plane*. Pengaturan jaringan dilakukan secara terpusat *logically centralized* oleh *controller*. ketika sebuah *controller* mengalami kelebihan *load* dan terjadi *Single Point of Failure* maka kinerja jaringan akan terganggu. *Software Defined Network* dapat mengatasi masalah tersebut dengan mengimplementasikan arsitektur *Multiple Distributed Controller* menggunakan metode *Dynamic Switch Migration*. Arsitektur *Multiple Distributed Controller* dalam penelitian ini menggunakan dua buah *controller* dengan peran *Master* dan *Slave*. Melalui simulasi menggunakan arsitektur *Multiple Distributed Controller* telah diuji kemampuan mekanisme *Dynamic Switch Migration* dalam menangani masalah kelebihan *load* pada *controller* dengan memindahkan sebagian *switch* dari *controller master* ke *controller slave* dan masalah *Single Point of Failure* dengan memindahkan seluruh *switch controller master* ke *Controller slave*.

Kata kunci: Software Defined Network, Dynamic Switch Migration, Multiple Distributed Controller, kelebihan load, controller slave, controller master

Abstract

Software Defined Network is a technology that can manage large-scale networks by separating control plane and data plane. Network settings are centrally logically centralized by the controller. when a controller is overloaded and a Single Point of Failure occurs, network performance will be disrupted. Software Defined Network can solve this problem by implementing the Multiple Distributed Controller architecture using the Dynamic Switch Migration method. The Multiple Distributed Controller architecture in this study uses two controllers with Master and Slave roles. Through simulation using the Multiple Distributed Controller architecture, the ability of the Dynamic Switch Migration mechanism to handle the problem of overloading the controller by moving a part of the switch to the slave controller and Single Point of Failure by moving all switches to the Slave Controller.

Keywords: Software Defined Network, Dynamic Switch Migration, Multiple Distributed Controller, overloaded, controller slave, controller master

1. Pendahuluan

Latar Belakang

Sebagai teknologi baru, *Software Defined Network* (SDN) memberi kemudahan dalam pengelolaan jaringan skala besar dengan memisahkan *control plane* dari *data plane* [9]. Akan tetapi kemampuan dari sebuah *single centralized controller* terbatas. sebagai contoh ketika *load* dari sebuah *controller* meningkat akan menyebabkan *controller* kelebihan *load* yang berdampak pada penurunan kinerja jaringan [4]. Disamping itu masalah ketika *controller* mengalami *failure* yang menyebabkan jaringan akan terputus. Dengan arsitektur *Multiple Distributed Controller* menggunakan mekanisme *Dynamic Switch Migration* pada *Software Defined Network* (SDN) dapat mengatasi pengelolaan perangkat jaringan semakin mudah [9].

Dynamic Switch Migration adalah pendekatan yang dapat menangani (Single Point of Failure) dan *load balancing* [1]. Dalam penerapannya, *Dynamic switch migration* memigrasikan *switch* ke *Controller slave* ketika *load* dari *controller master* melebihi kapasitasnya [1].

Dengan adanya mekanisme *Dynamic Switch Migration* yang diterapkan pada arsitektur *Multiple Distributed Controller* dapat mencegah terjadinya kelebihan *load controller* dan *failure* karena ketersediaan *controller* untuk mem-backup *controller* lainnya tetap terjaga, sehingga kinerja jaringan tidak terganggu.

Topik dan Batasannya

Pada jaringan SDN terdapat desain arsitektur *Multiple Distributed Controller* yang memungkinkan sebuah jaringan SDN memiliki lebih dari satu *controller*. Oleh sebab itu pada penelitian ini akan digunakan dua buah *controller* dengan jenis *Floodlight* [3] sebagai *control plane* yang ditempatkan satu level/ *Flat Architecture* agar memiliki kemampuan dan tanggung jawab yang setara. sedangkan untuk bagian *Data plane* menggunakan mininet sebagai emulator yang dapat mensimulasikan *Open Flow* switch.

Penelitian ini akan melakukan perancangan menggunakan topologi *Fat tree* yang terdiri dari dua buah *controller* dan empat buah *switch* yang masing-masing terhubung ke *controller*. Permasalahan yang ditangani berupa kelebihan *load* pada *controller* yang menyebabkan penurunan kinerja sebuah *controller* dan *Single Point of Failure* yaitu keadaan dimana sebuah *controller* tidak berfungsi akibat dilakukan *shutdown*. Penelitian ini akan menganalisis nilai *throughput* yaitu kemampuan *controller* dalam menangani *response/sec* sebelum dan sesudah dilakukan *switch migration* serta *completion time* yaitu waktu yang dibutuhkan untuk melakukan *switch migration* dari *controller master* ke *controller slave*.

Tujuan

Penelitian ini bertujuan untuk merancang arsitektur *Multiple Distributed Controller* pada jaringan SDN dengan menerapkan mekanisme *Dynamic Switch Migration* untuk mencegah permasalahan kelebihan *load* pada *controller* dan *Single Point of Failure* agar fungsi dari sebuah *controller* pada sebuah jaringan SDN tetap terjaga. Disamping itu dapat diketahui pengaruh arsitektur *Multiple Distributed Controller* terhadap performansi jaringan SDN dengan parameter *throughput* dan *Completion Time* saat terjadi *switch migration*.

Organisasi Tulisan

Pada bagian selanjutnya terdiri dari studi terkait, sistem yang dibangun, evaluasi dan kesimpulan. Pada bagian studi terkait akan dijelaskan studi literatur dan teori dasar mengenai topik yang diangkat pada penelitian ini. Penjelasan mengenai desain arsitektur dan implementasi sistem yang dibangun akan dijelaskan lebih lanjut pada bagian sistem yang dibangun. Hasil dari implementasi akan dianalisis pada bagian evaluasi. kemudian hasil dari uraian yang didapat dari penelitian akan ditampilkan pada bagian kesimpulan.

2. Studi Terkait

Pada penelitian sebelumnya, *Dynamic Switch Migration* pada *Multiple Distributed Controller* digunakan untuk menangani *load balancing* menggunakan metode *load informing strategy* [8]. Metode ini diterapkan pada *controller* dengan empat komponen utama yakni, *load measurment*, *load informing*, *balance decision* dan *switch migration*. metode ini memungkinkan untuk setiap *controller* berbagi data *load information* satu sama lain. Apabila ditemukan *controller* dengan beban trafik yang *overload* akan dilakukan *balance descision* yang mana menentukan *controller* dengan beban terberat, menentukan *switch* yang akan dipindahkan serta menentukan target *switch*. setelah komponen *balance descision* dilakukan, maka akan dijalankan *switch migration* komponen yang mana memindahkan beban trafik *switch* ke *controller* dan *switch* yang menjadi target.

Pada penelitian [9], *Dynamic Switch Migration* di implementasikan menggunakan database untuk menyimpan *load information* dan status dari setiap *controller*. Database akan melakukan *update* data setiap saat untuk menyesuaikan fluktuasi data di setiap *controller*. Algoritma *load balancing and failure recovery* akan melakukan *load* data status *controller* dan *load information* dari database. apabila terdeteksi bahwa beban pada *switch overload*, maka algoritma *load balancing and failure recovery* akan dijalankan untuk memindahkan beban trafik ke *controller* yang memiliki beban trafik paling minimal.

Pada penelitian [10], DALB adalah metode yang memungkinkan setiap *controller* dapat melakukan *load balancing decision* secara lokal. Jika ditemukan *overloaded controller* maka akan mekanisme *balance decision* akan dijalankan. Sebelum mekanisme *balance decision* dijalankan secara lokal, maka akan data *load information* dari seluruh *controller* akan dikumpulkan terlebih dahulu sebagai informasi ketika akan dilakukan mekanisme *balance decision*.

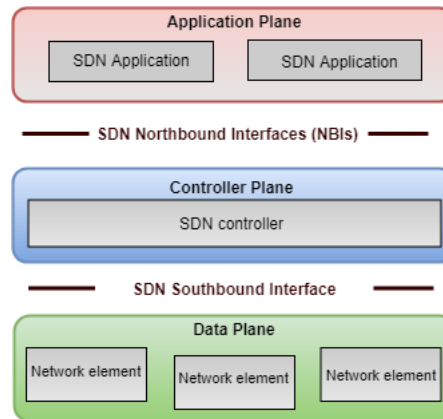
Dynamic Switch Migration

Dynamic Switch Migration adalah pendekatan yang dapat menangani (Single Point of Failure) dan *load balancing* [1]. *Dynamic Switch Migration* dapat diimplementasikan ke dalam tiga studi kasus. Pertama, jika beban trafik melampaui kapasitas semua *controller*, maka ada penambahan *controller* baru dan beban trafik *switch* dipindahkan ke *controller* tersebut [7]. kedua, jika *controller* dimatikan atau *sleep* maka *switch* dimigrasikan [7]. ketiga, jika *load controller overload* dan tidak ada penambahan *controller*, maka harus memigrasikan *switch* dengan beban

trafik tinggi ke *controller* lain yang bebannya tidak *overload* [7].

Software Defined Network

Software Defined Network adalah suatu paradigma jaringan yang memisahkan antara *control plane* dan *data plane* [1]. SDN menggunakan *Centralized Control Plane* dan *Distributed Forwarding Data Plane*, dua bagian yang saling terpisah satu sama lain [9]. Protokol antarmuka pada SDN memungkinkan SDN dapat di program. Protokol antarmuka *northbound* menyediakan interaksi antara *Application plane* dengan *Control plane*, sedangkan protokol antarmuka *Southbound* menyediakan interaksi antara *Control plane* dengan *Data plane* [9].



Gambar 1. Arsitektur *Software defined network* [3]

Load Balancing

Load Balancing adalah metode yang digunakan untuk pembagian beban trafik data dalam jaringan. metode ini mendistribusikan beban trafik pada dua atau lebih jalur koneksi dengan seimbang agar trafik dapat berjalan secara optimal. Selain itu *load balancing* membagi beban secara merata pada sebuah sistem jaringan atau sumber daya lainnya untuk mendapatkan pemanfaatan *resource* dan *performansi* yang maksimal.

Open Flow Protocol

Open Flow Protocol adalah protokol yang menjadi standar komunikasi pada *Software Defined Network*. *Open Flow* menjadi protokol komunikasi yang memungkinkan interaksi antara *control plane* dengan *data plane* secara baik [6].

3. Sistem yang Dibangun

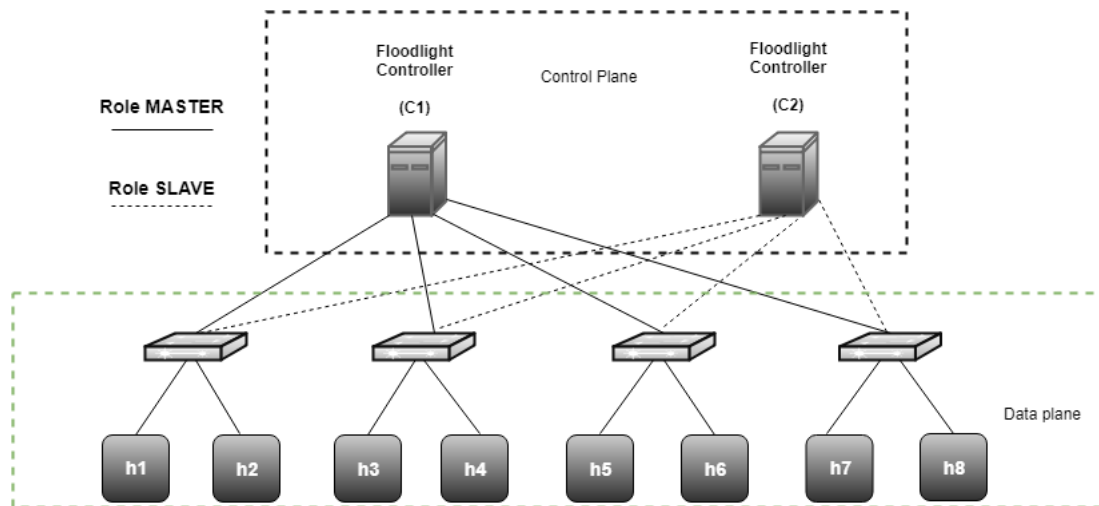
3.1 Perancangan Arsitektur Sistem

Perancangan arsitektur *Multiple Distributed Controller* terdiri dari dua bagian yaitu perancangan topologi dan perancangan *software*. Perancangan topologi pada penelitian ini menggunakan topologi *Fat tree* yang terdiri dari 2 buah *controller* dan empat buah *Open Flow Switch*. Masing-masing *switch* akan terhubung pada kedua *controller* dengan *role* yang berbeda. Ilustrasi topologi *Multiple Distributed Controller* dapat dilihat pada gambar 2

berdasarkan topologi diatas C1 bertindak sebagai *controller master* dan C2 sebagai *controller slave*. C1 sebagai *controller master* memiliki tanggung jawab penuh terhadap jaringan dengan status *ACTIVE* dan C2 sebagai *controller slave* akan menjadi *backup*. Apabila C1 mengalami kelebihan *load* yang ditandai dengan banyaknya *Packet in* yang diterima maka C2 akan menghandel beberapa *switch* dengan mengubah peran dari *ROLE SLAVE* menjadi *ROLE MASTER*, dan apabila C1 mengalami *failure* maka C2 akan mengubah peran seluruh *switch* dari *ROLE SLAVE* menjadi *ROLE MASTER* sehingga kinerja jaringan tidak terputus.

3.2 Load collection

Load collection adalah modul yang dirancang untuk menangani *arrival packet-in rate*. Modul ini akan menghitung setiap *packet-in* yang datang ke *controller* dan menjadikannya sebagai *load* dari sebuah *controller*. Modul ini hanya akan berjalan pada *controller* dengan peran *ROLE MASTER* terhadap *switch* yang terhubung.



Gambar 2. Topologi Multiple Distributed Controller

3.3 Switch Migration

Switch Migration adalah mekanisme yang dijalankan melalui *REST API Service* yang disediakan oleh *floodlight* [3]. *Switch* dengan peran *ROLE MASTER* yang terhubung pada *controller master* dan peran *ROLE SLAVE* yang terhubung pada *controller slave* akan dibalik perannya pada masing-masing *controller*. Mekanisme ini dilakukan agar kedua *controller* dapat bekerja sama dalam meng-handle *load* (arrival packet in rate) dari setiap *switch*.

3.4 Spesifikasi Software

Berdasarkan rancangan topologi dan mekanisme yang telah ditentukan, maka diperlukan software untuk mendukung rancangan tersebut. Berikut adalah *software* yang digunakan:

1. Ubuntu 16.04 LTS, sebagai sistem operasi.
2. Floodlight 1.2 sebagai controller
3. Mininet sebagai emulator untuk mensimulasikan jaringan
4. Eclipse sebagai *tools* untuk memprogram Floodlight.
5. Cbench sebagai *Controller Benchmarking* untuk menganalisis performansi *controller*

3.5 Skenario Pengujian

Skenario Pengujian Throughput

Skenario pengujian ini dilakukan dengan menjalankan kedua *controller* yang terhubung pada emulator *mininet* [5]. Dengan menggunakan *Cbench* [2] sebagai *tools* untuk *benchmarking controller*, maka *controller* akan dites dengan konfigurasi 16 *switch*, 1000 *MAC address* serta dijalankan dalam mode *throughput*. Pengujian ini dijalankan sebelum dan sesudah *controller* melakukan *switch migration* untuk mengetahui perbandingan nilai *throughput* antara *single controller* dan *double Controller*.

Skenario Pengujian Completion Time

Skenario pengujian ini dilakukan dengan menjalankan kedua *controller* yang terhubung pada emulator *mininet* dengan konfigurasi sesuai arsitektur yang telah dirancang. *Controller* dibiarkan *running* selama 45 detik. Pada saat detik ke-10 ketika *controller* menerima peningkatan jumlah *packet-in* secara signifikan maka dilakukan *switch migration* dengan menggunakan *Rest API Service* seperti pada gambar 3 dan gambar 4

pergantian *role* dilakukan pada dua buah *switch* sehingga masing-masing *controller* C1 dan C2 akan meng-handle dua buah *switch* dengan *ROLE MASTER*. Setelah dilakukannya skenario ini, kini *controller slave* atau C2 ikut meng-handle *switch* yang terhubung dengan peran *ROLE MASTER*.

Skenario Pengujian Single Point of Failure

Pada pengujian ini kedua *controller* telah berjalan dan terhubung pada *mininet* sesuai dengan arsitektur yang sudah dirancang. Untuk mengetahui mekanisme *Dynamic Switch migration* berjalan ketika *controller* mengalami *failure* dan memastikan kinerja jaringan tidak terputus maka *controller* C1 di-*shutdown* dengan perintah (Ctrl+C).

```

blackbox@blackbox:~/oflops$ curl http://127.0.0.1:8082/wm/core/switch/"00:00:00:00:00:00:00:01"/role/json -X POST -d '{"role": "MASTER"}' |python -m json.tool
{
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
  Dload    Upload   Total     Spent    Left     Speed
100    56    0    36  100    20    451    251  --:--:--  --:--:--  --:--:--  455
{
  "00:00:00:00:00:00:00:01": "MASTER"
}
blackbox@blackbox:~/oflops$ curl http://127.0.0.1:8082/wm/core/switch/"00:00:00:00:00:00:00:02"/role/json -X POST -d '{"role": "MASTER"}' |python -m json.tool
{
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
  Dload    Upload   Total     Spent    Left     Speed
100    56    0    36  100    20    1725    958  --:--:--  --:--:--  --:--:--  1800
{
  "00:00:00:00:00:00:00:02": "MASTER"
}
    
```

Gambar 3. Perintah pergantian Role dari Master ke Slave

```

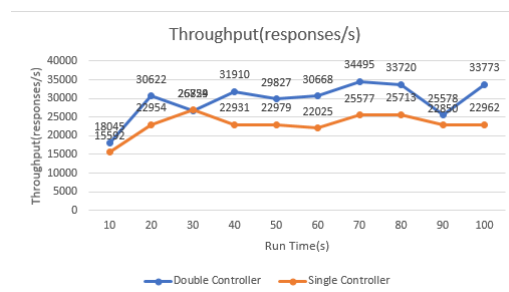
blackbox@blackbox:~/oflops$ curl http://127.0.0.1:8080/wm/core/switch/"00:00:00:00:00:00:00:01"/role/json -X POST -d '{"role": "SLAVE"}' |python -m json.tool
{
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
  Dload    Upload   Total     Spent    Left     Speed
100    54    0    35  100    19    325    176  --:--:--  --:--:--  --:--:--  327
{
  "00:00:00:00:00:00:00:01": "SLAVE"
}
blackbox@blackbox:~/oflops$ curl http://127.0.0.1:8080/wm/core/switch/"00:00:00:00:00:00:00:02"/role/json -X POST -d '{"role": "SLAVE"}' |python -m json.tool
{
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
  Dload    Upload   Total     Spent    Left     Speed
100    54    0    35  100    19    2143    1163  --:--:--  --:--:--  --:--:--  2187
{
  "00:00:00:00:00:00:00:02": "SLAVE"
}
    
```

Gambar 4. Perintah pergantian Role dari Slave ke Master

4. Evaluasi

4.1 Hasil dan Analisis Uji Throughput

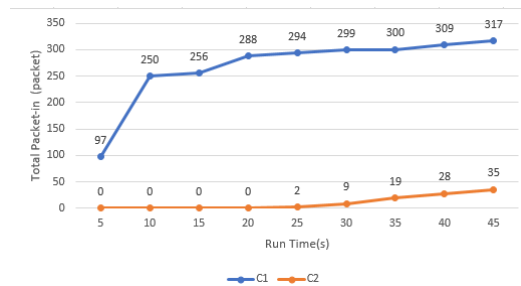
Berdasarkan hasil dari skenario pengujian *Throughput* yang dilakukan selama 100 detik terhadap *single controller* dan *double controller* menghasilkan nilai *throughput* yang berbeda. Berdasarkan grafik pada gambar 5, nilai *throughput* dari *double controller* lebih besar jika dibandingkan dengan *single controller*. rata-rata *responses/s* yang dapat diterima oleh *double controller* sebesar 30.817 *responses/s* sedangkan *single controller* hanya dapat menerima 23.848 *responses/s*. Hal ini terjadi karena *double controller* membagi tugas dalam mengatur kinerja *switch*. seperti yang kita ketahui, *switch* adalah perangkat yang mengirimkan *packet in* kepada *controller*. ketika sebagian *switch* dimigrasikan ke *controller slave* maka beban dari *controller master* tidak terlalu berat yang mana dalam hal ini dapat merespon *packet in* yang lebih banyak.



Gambar 5. Uji Throughput Controller

4.2 Hasil dan Analisis Uji Completion Time

Evaluasi terhadap hasil dari uji *completion time* adalah untuk mengetahui seberapa lama *controller slave/C2* dapat merespon *packet in* ketika terjadi *switch migration*. Seperti yang kita ketahui, jumlah *Packet in* yang diterima oleh *controller* adalah nilai *load* dari *controller* tersebut. Berdasarkan grafik pada gambar 6, dapat dilihat bahwa pada detik ke-10 terjadi peningkatan jumlah *packet in* secara signifikan yang menyebabkan harus dilakukannya *switch migration*. *switch migration* dilakukan pada detik ke-10 dengan memigrasikan dua buah *switch* ke *controller slave/C2* dari *controller master/C1*. terlihat pada detik ke-27 *controller slave/C2* mulai menerima *packet-in* yang menunjukkan bahwa *controller slave/C2* sudah meng-handle setiap aktivitas *switch*. Waktu yang dibutuhkan sejak dilakukannya *switch migration* terhadap *controller slave/C2* menerima *packet in* adalah 17 detik dan hingga detik ke-45 kedua *controller* menerima *packet-in* secara bersamaan. Hal ini menunjukkan adanya pembagian penanganan *load* pada kedua *controller* tersebut.



Gambar 6. Uji Completion Time

4.3 Hasil dan Analisis Uji Fungsionalitas SPoF

Pada pengujian ini kedua *controller* berjalan dan terhubung dengan *mininet* sesuai dengan arsitektur yang telah ditentukan. *controller master/C1* dan *controller slave/C2* terhubung masing-masing pada empat buah *switch* dengan peran *ROLE MASTER* dan *ROLE SLAVE*. Berdasarkan *log controller slave/C2* pada gambar 7, dapat dilihat bahwa seluruh *switch* dari *controller master/C1* telah dimigrasikan ke *controller slave/C2*. Hal ini terjadi berdasarkan skenario yang dijalankan pada pengujian *Single Point of Failure* yang mana *controller master/C1* di-*shutdown* sehingga mekanisme *Dynamic Switch Migration* dijalankan agar kinerja jaringan tidak terputus.

```

2018-08-22 21:03:38.536 INFO [n.f.simpleleft.FT] DEFINED 00:00:00:00:00:00:01
as ROLE_MASTER, reply.getRole:ROLE_MASTER!
2018-08-22 21:03:38.536 INFO [n.f.c.i.OFSwitchHandshakeHandler] Clearing flow t
ables of 00:00:00:00:00:00:00:01 on upcoming transition to MASTER.
2018-08-22 21:03:38.537 INFO [n.f.simpleleft.FT] DEFINED 00:00:00:00:00:00:02
as ROLE_MASTER, reply.getRole:ROLE_MASTER!
2018-08-22 21:03:38.537 INFO [n.f.c.i.OFSwitchHandshakeHandler] Clearing flow t
ables of 00:00:00:00:00:00:00:02 on upcoming transition to MASTER.
2018-08-22 21:03:38.537 INFO [n.f.c.i.OFSwitchHandshakeHandler] Clearing flow t
ables of 00:00:00:00:00:00:00:03 on upcoming transition to MASTER.
2018-08-22 21:03:38.538 INFO [n.f.simpleleft.FT] DEFINED 00:00:00:00:00:00:03
as ROLE_MASTER, reply.getRole:ROLE_MASTER!
2018-08-22 21:03:38.539 INFO [n.f.c.i.OFSwitchHandshakeHandler] Clearing flow t
ables of 00:00:00:00:00:00:00:04 on upcoming transition to MASTER.
2018-08-22 21:03:38.539 INFO [n.f.simpleleft.FT] DEFINED 00:00:00:00:00:00:04
as ROLE_MASTER, reply.getRole:ROLE_MASTER!

```

Gambar 7. Log Controller Slave saat terjadi Switch Migration

5. Kesimpulan

Berdasarkan dari hasil pengujian terhadap desain arsitektur *Multiple Distributed Controller* menggunakan meka-nisme *Dynamic Switch Migration* dengan melakukan uji *throughput*, *Completion time* dan uji fungsionalitas untuk mengatasi masalah *Single Point of Failure*, penggunaan arsitektur *Multiple Distributed Controller* mempengaruhi performansi dengan menghasilkan nilai *throughput* yang lebih besar yaitu 30.817 responses/s jika dibandingkan dengan *single controller*. Hal ini menandakan desain arsitektur ini dapat menangani masalah kelebihan *load* pada *controller*. Disamping itu lamanya waktu yang dibutuhkan/ *completion time* untuk memigrasikan *switch* hanya

sekitar 17 detik. Dan desain arsitektur ini dapat mengatasi masalah *Single Point of Failure* dengan memigrasikan seluruh *switch* dari *controller master/C1* ke *controller slave/C2* sehingga kinerja jaringan tidak terganggu bahkan terputus. Untuk penelitian selanjutnya yang berkaitan dengan arsitektur *Multiple Distributed Controller* menggunakan mekanisme *Dynamic Switch Migration* adalah menambahkan mekanisme/metode yang dapat meng-handle masalah kelebihan *load* dan *Single Point of Failure* untuk *controller* dengan jumlah lebih dari dua buah.

Daftar Pustaka

- [1] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhanian, R. Ahmed, and R. Boutaba. Dynamic controller provisioning in software dented network. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 18–25, Oct 2013.
- [2] Cbench. <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343657/cbench+new>. Accessed: May 08,2018.
- [3] Floodlight. <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343542>. Accessed: March 10,2018.
- [4] W. Lan, F. Li, X. Liu, and Y. Qiu. A load balancing strategy of sdn controller based on distributed decision. In *2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, pages 259 – 262, April 2018.
- [5] Mininet. <https://mininet.org>. Accessed: March 20, 2018.
- [6] OpenFlow. <https://www.opennetworking.org/sdn-definition/>. Accessed: December 10,2017.
- [7] C. Wang, B. Hu, S. Chen, D. Li, and B. Liu. A switch migration-based decision-making scheme for balancing load in sdn. In *IEEE Access (Volume: 5)*, pages 4537 – 4544, March 2017.
- [8] J. Yu, Y. wang, K. Pei, S. Zhang, and J. Li. A load balancing mechanism for multiple sdn controllers based on load informing strategy. In *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4, Nov 2016.
- [9] H. Zhong, J. Sheng, J. Cui, and Y. Xu. Scplbs: A smart cooperative platform for load balancing and security on sdn distributed controllers. In *Cybernetics (CYBCONF), 2017 3rd IEEE International Conference on*, pages 1–6, July 2017.
- [10] Y. Zhou, M. Zhu, L. Xiao, L. Ruan, W. Duan, D. Li, R. Liu, and M. Zhu. A load balancing strategy of sdn controller based on distributed decision. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, pages 851 – 856, sept 2014.