Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 2, No. 10, Oktober 2018, hlm. 3968-3975 http://j-ptiik.ub.ac.id

Implementasi *Link Fast-Failover* Pada *Multipath Routing* Jaringan *Software-Defined Network*

Muhammad Aji Wibowo[1], Widhi Yahya[2], Dany Primanita Kartikasari[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]muhammadaji01@gmail.com, [2]widhi.yahya@ub.ac.id, [3]dany.jalin@ub.ac.id

Abstrak

Software-defined network merupakan sebuah konsep pendekatan jaringan komputer dimana pengkontrol (*control plane*) dipisahkan dengan *data plane*. Komunikasi antara pengkontrol dangan *data plane* menggunakan protokol *OpenFlow*. Dalam sebuah arsitektur SDN sering terjadi mengalami gangguan seperti *failure link*. Pada protokol *openflow* terbaru terdapat sebuah tipe grup yang mengatasi gangguan pada link tersebut yaitu menggunakan mekanisme *fast-failover*. Pengujian yang dilakukan untuk mengukur kinerja dari sistem meliputi pengujian fungsionalitas, *response time* dan *packet loss*. Hasil dari pengujian yang didapat dari penelitian ini mampu melakukan pencarian jalur pada saat sebelum *link down* dan setelah *link down*. Pada saat *link 1* di *down* dalam topologi memiliki *response time* dengan rata-rata 0,38-0,681 ms dan di *link 2* memiliki rata-rata 0,272 – 0,549 ms. Ketika *link* di *down*, paket yang hilang sampai menemukan jalur yang baru dapat diketahui nilai *packet loss*. *packet* loss yang didapat dari simulasi mempunyai rata-rata 0,423%-0,636%.

Kata kunci: *Fast-failover. Multipath Routing, Software-Defined Network, Openflow*

Abstract

Software-defined network is a concept of computer network approach where the control plane is separated by data plane. The communication between the data plane controller uses the OpenFlow protocol. In an SDN architecture it often happens to have interference such as failure link. in the latest openflow protocol there is a group type that overcomes the interruption of the link is using a fast-failover mechanism. Tests conducted to measure the performance of the system include testing functionality, response time and packet loss. The results of the test obtained from this study are able to search the path at the time before the link down and after the link down. At the time of link 1 in down in topology has a response time with an average of 0.38-0.681 ms and in link 2 has an average of 0.272 - 0.549 ms. When the link is down, the lost packet until it finds the new path can be known the value of packet loss. packet loss obtained from the simulation has an average of 0.423% -0.636%.

Keywords: *Fast-failover. Multipath Routing, Software-Defined Network, Openflow*

1. PENDAHULUAN

Software-Defined Network (SDN) sebuah konsep pendekatan jaringan komputer dimana sistem pengkontrol dari arus data dipisahkan dari perangkat lainnya. menurut (Antonio Capone, Carmelo Cascone, Alessandro Q.T. Nguyen, & Brunilde Sanso, 2015), pendekatan jaringan secara traditional diganti dengan jaringan terpusat yang dapat mengatur *traffic* dengan melalui aturan pemrograman *forwading* yang tingkat rendah ke dalam *switch* berdasarkan abstraksi yang sederhana dari fungsi *switch* seperti didefinisikan di *OpenFlow* dengan *flow table*. Meskipun SDN dan *OpenFlow* menyediakan fleksibilitas yang besar dan sebuah *platform* yang sangat kuat untuk memprogram beberapa tipe aplikasi jaringan yang inovatif tanpa keterbatasan yang dialami protokol-protokol terdistribusi, untuk mengimplementasi fungsi-fungsi tradisional penting pada SDN dan *OpenFlow*, seperti ketahanan kegagalan, tidak mudah maupun efesien, karena reaksi terhadap event atau kejadian pada jaringan akan selalu melibatkan *controller* pusat (pemberitahuan dan instalasi aturan baru forwading) dengan *delay* dan beban sinyal yang tidak dapat dihindari (Antonio Capone, Carmelo Cascone, Alessandro

Q.T. Nguyen, & Brunilde Sanso, 2015). Dalam SDN sering terjadi sebuah gangguan link yang mengganggu pada *traffic*.

Link failure atau kegagalan *link* pada suatu jaringan merupakan salah satu permasalahan yang perlu diperhatikan oleh operator jaringan. Efek dari *link failure* yaitu *packet loss* yang akan menyebabkan pengurangan *throughput* pada jaringan sebenarnya (Ying-Dar Lin, Hung-Yi Teng, Chia-Rong Hsu, Chun-Chieh Liao, & Yuan-Cheng Lai, 2016). Untuk mengatasi hal tersebut diperlukan suatu mekanisme manejemen kegagalan sehingga layanan jaringan dapat disediakan dengan lancar dan dengan gangguan yang minimal. Tantangannya adalah dibutuhkan reaksi yang cepat terhadap kegagalan dari komponen jaringan dengan mengkonfigurasi ulang alokasi sumber daya agar dapat memanfaatkan infrastruktur yang bertahan untuk menyediakan layanan (Antonio Capone, Carmelo Cascone, Alessandro Q.T. Nguyen, & Brunilde Sanso, 2015).

Pada versi *openflow* terbaru yaitu *openflow* 1.3 memperkenalkan fitur terbaru pada *OpenFlow* yaitu mekanisme *fast-failover*, untuk memungkinkan reaksi yang local dan cepat terhadap kegagalan tanpa perlu meminta keputusan dari *controller* terpusat. Ini diperoleh dengan memasang beberapa tindakan terhadap *entry flow* yang sama dan menerapkannya berdasarkan status dari *link* (aktif atau gagal) (Antonio Capone, Carmelo Cascone, Alessandro Q.T. Nguyen, & Brunilde Sanso, 2015). Pada mekanisme *Fast-failover*, *controller* secara periodik memperhitungkan *multiple path* secara berkala untuk *source-destination* dan memasang *flow* dan *gorup entry* pada masing-masing switch. Ketika kesalahan *link* terdeteksi, *switch* dapat mengalihkan jalur ke jalur yang lain (Ying-Dar Lin, Hung-Yi Teng, Chia-Rong Hsu, Chun-Chieh Liao, & Yuan-Cheng Lai, 2016). Koneksi dari *host* dapat menata ulang *link* yang rusak ketika jalur sudah dialihkan ke jalur yang lain untuk sementara. Dengan memanfaatkan *multipath routing*, *controller* menghitung jalur yang mungkin sebagai pohon akar yang terpisah antara masing-masing *source-destination* (Ghannami & Shao, 2016).

Routing adalah proses menentukan rute atau jalur yang diambil oleh paket dimana mereka mengalir dari pengirim ke penerima. Algoritme yang menghitung jalur ini disebut algoritme *routing*. Sebuah algoritme *routing* akan menentukan, Misalnya, jalan di sepanjang dimana paket mengalir dari H1 ke H2 (Kurose & Ross, 2013). Algoritme *routing* yang digunakan adalah algoritme *Dijkstra* dan algoritme DFS. Algoritme *Link-State* atau yang biasa disebut dengan algoritme *Dijkstra*. Dalam algoritme *link-state*, topologi jaringan dan bobot semua *link* diketahui oleh setiap *node*, topologi dan bobot link tersebut kemudian akan diproses sebagai masukan untuk algoritme LS. Dalam prakteknya, hal ini dicapai dengan setiap node melakukan *broadcast* yang berisi paket *Link-State* dimana paket tersebut berisi bobot *link* yang terhubung dan identitas masing-masing node. Sedangkan Algoritme DFS yaitu metode pencarian jalur yang dilakukan dengan menggunakan struktur data *stack* yang dapat menyimpan rute kemudian melakukan ekspansi lagi meskipun lagi sudah ditemukan satu jalur ke tujuan (Maulana S., Yahya, & Rizqika A., 2017).

Dalam penelitian, penulis telah melakukan implementasi *Fast-failover Link* pada *multipath routing* jaringan *OpenFlow software-defined network* yang berdasarkan pada gangguan di *link* topologi dan penelitian ini diimplementasikan pada Mininet dan memakai *Ryu Controller* yang menggunakan Bahasa *Phyton*. Dengan itu diharapkan dapat mengatasi *link* seperti *link* yang putus pada topologi.

2. DASAR TEORI

2.1 *Software-Defined Network*

Software-Defined Network (SDN) adalah arsitektur yang muncul yang bersifat dinamis, dapat di kelola, hemat biaya dan mudah beradaptasi, sehingga ideal untuk *high-bandwidth*, sifat dinamis dari aplikasi saat ini. Arsitektur ini memisahkan *network control* dan fungsi *forwading* yang memungkinkan control jaringan langsung diprogram dan dasar infrasturktur yang abtraksi untuk aplikasi dan layanan jaringan. (Open Networking Foundation, n.d.).

2.2 *OpenFlow*

OpenFlow adalah antarmuka komunikasi standar pertama kali didenifisikan antara control dan *forwading* lapisan asitektur SDN. *OpenFlow* memungkinkan akses langsung ke dan manipulasi *forwading* bidang perangkat jaringan seperti *switch* dan *router*, baik fisik dan virtual

(berdasarkan *hypervisor*) (Open Networking Foundation, n.d.).

2.3 Mekanime *Fast-failover*

Mekanisme *failover* merupakan suatu teknik pada jaringan dengan memberikan dua jalur koneksi (primary dan backup link) atau lebih dimana ketika jalur utama mati, maka koneksi akan tetap berjalan dengan mengalihkan ke backup link. prosedur mekanisme failover menentukan kelangsungan operasional jaringan. Mekanisme failover dapat dirancang sehingga dapat segera mungkin untuk bertindak ketika ada gangguan yang muncul (Purwiadi, et al., 2018).

2.4 *Controller*

Controller SDN adalah sebuah aplikasi di SDN yang mengelolal *flow control* untuk mengaktifkan jaringan cerdas. *Controller* SDN didasarkan pada protocol, seperti *OpenFlow*, yang memungkinkan server untuk memberitahu *Switch* dimana untuk mengirmkan paket (Rouse, 2012).

2.5 *Ryu Controller*

Ryu dalam Bahasa jepang "flow/aliran". Ryu adalah komponen berdasarkan SDN Framework. Ryu menyediakan komponen perangkat lunak dengan API didefinisikan dengan baik yang membuatnya mudah untuk pengembang untuk membuat manajemen jaringan baru dan control aplikasi. Pengembangan aplikasi untuk ryu dapat dilakukan dengan menggunakan Bahasa *python* atau dengan mengirmkan pesan JSON melalui API yang tersedia. Ryu mendukung berbagai protocol untuk mengelolal perangkat jaringan seperti *OpenFlow*, Netconf, OF-config dan lain-lain. Ryu mendukung penuh *OpenFlow* versi 1.0, 1.2, 1.3, 1.4, 1.5 dan Nicira Extension. Semua kode tersedia secara bebas di bawah lisensi Apache 2.0. (RYU SDN Framework Community, 2014)

2.6 *Multipath Routing*

Multipath adalah teknik *routing* yang menggunakan beberapa jalur alternatif melalui jaringan, yang dapat menghasilkan berbagai manfaat seperti toleransi kegagalan, meningkatkan bandwidth atau meningkatkan keamanan. *Multipath Routing* telah diekslporasi dalam beberapa konteks yang berbeda. Jaringan sirkui tradisional ada telepon menggunakan tipe dari *Multipath Routing* yang disebut *alternate path routing.* Di *alternate path routing*, tiap sumber node dan node tujuan mempunyai beberapa bagian yang terdiri dari jalur utama dan jalur alternatif (Mueller, et al., 2003).

2.7 Algoritme *Depth-First Search* (DFS)

Algoritme DFS adalah salah satu algoritme yang digunakan untuk pencarian jalur (pip, 2015). Pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri, jika pada level yang paling dalam, solusi belum ditemukan, maka pencarian dilanjutkan pada node sebelah kanan. Node yang kiri dapat dihapus dari memori. Jika pada level yang paling dalam ditemukan solusi, maka pencarian dilanjutkan pada level sebelumnua. Untuk melakukan ini dengan benar kita perlu untuk menjaga jalur yang telah dikunjung ditambah bagaimana kita harus (jalur ke…) dimana kita berada saat ini, sehingga bisa kembali pada jalur sebelumnya (Heap, 2002).

```
DFS(G,v)  ( v is the vertex where the search starts )
       Stack S := {}; ( start with an empty stack )
       for each vertex u, set visited[u] := false;
       push S, v;
       while (S is not empty) do
          u := pop S;
          if (not visited[u]) then
            visited[u] := true;
            for each unvisited neighbour w of u
                push S, w;
          end if
       end while
END DFS()
```

Gambar 1 *pseudocode* Algoritme DFS

2.8 Mininet

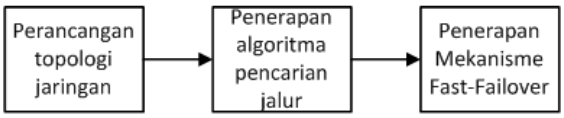
Mininet adalah emulator jaringan yang realistis, menjalankan kernel asli, *switch* dan aplikasi code pada satu mesin (VM, cloud atau *native*), dalam hitungan detik, dengan satu (Mininet team, 2012).

3. PERANCANGAN DAN ANALISIS

3.1 Perancangan sistem

Perancangan sistem memuat langkah-langkah perencanaan dan perancangan atas suatu sistem yang dapat memenuhi kebutuhan berdasarkan analisis kebutuhan berdasarkan analisis kebutuhan yang telah dilakukan. Untuk memenuhi kebutuhan tersebut, yaitu menghasilkan sistem jaringan yang dapat melakukan *fast-failover* dengan algoritme pencarian jalur DFS, langkah- langkah perancangan yang dilakukan dapat dilihat pada

gambar 2.



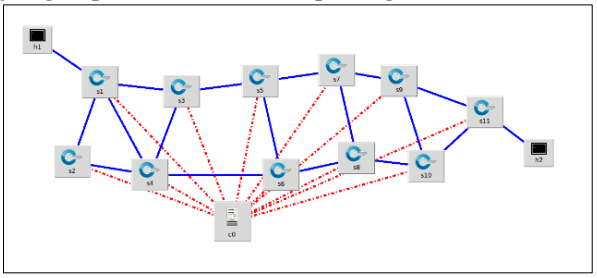
Gambar 2 proses perancangan sistem

Bedasarkan gambar 2 perancangan sistem ini terdiri tiga tahapan, yang dapat dijelaskan sebagai berikut.

1. Perancangan topologi jaringan, memuat proses perancangan topologi atau skema jaringan yang cocok untuk permasalahan yang dilakukan, yaitu multipath routing pada topologi dengan beberapa jalur
2. Penerapan algoritme pencarian jalur, memuat proses untuk menerapkan algoritme pencarian jalur pada perancangan topologi. Untuk penacarian jalur ditentukan dengan jalur yang terpendek yang ditemukan oleh pencarian jalur yang digunakan.
3. Penerapan mekanisme *Fast-failove*r, memuat proses untuk menerapkan mekanisme f*ast-failover* pada sistem yang digunakan. Penerapan Mekanisme *fast-failover* yaitu untuk mengatasi masalah pada sebuah link pada perancangan topologi.

3.2 Perancangan topologi

Untuk dapat memanfaatkan *Fast-failover* pada jaringan SDN, diperlukan suatu topologi yang memiliki beberapa jalur yang redundan yang dapat dilalui suatu node jaringan. Jalur yang redundan dapat didefinisikan dengan: untuk suatu set pasangan sumber dan tujuan pada komunikasi jaringan, terdapat beberapa jalur yang dapat dilalui oleh set pasangan tersebut.



Gambar 3 topologi pada mininet

Pada gambar 3 topologi yang digunakan untuk sistem ini. Topologi ini terdiri dari 11 *Switch*, 2 *host* dan 1 *controller*. Masing-masing link yang menyambungkan antar switch diberikan *bandwidth* sebesar 1 Ghz.

3.3 Simulasi dan Analisis

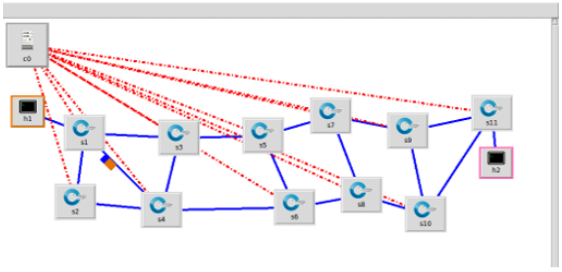
pengujian ini dilakukan dengan membandingkan mekanisme *fast-failover* dengan mekanisme *fail path (recovery)*. Terdapat 3 pengujian yaiu pengujian fungsionalitas, pengujian *response time* dan pengujian *packet loss*.

1. Pengujian Fungsionalitas

Pengujian fungsionalitas ini akan menjelaskan bagaimana mekanisme *fast-failover* dan *fail path* bekerja. Berikut pengujian fast-failover dan mekanisme *fail path (recovery)*. Pada pengujian ini diemulasikan oleh emulator miniNAM. Fungsi dari emulator miniNAM yaitu dapat melihat paket-paket yang dilewati. Untuk melihat paket ICMP yaitu menggunakan PING. Paket tersebut dapat terlihat pada emulator MiniNAM.

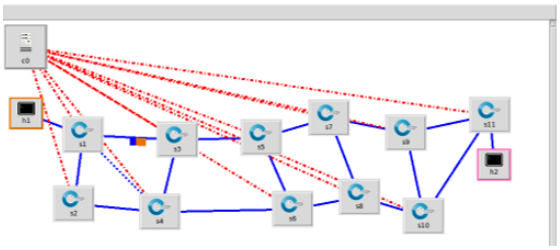
a. Mekanisme *Fast-Failover* dengan pencarian jalur *Algoritme DFS*

Pada pengujian ini dijelaskan bagaimana pengujian mekanisme *fast-failover* berjalan dengan menggunakan algoritme DFS.



Gambar 4 topologi dengan tidak ada gangguan (DFS)

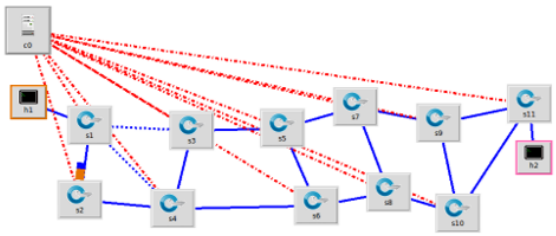
Pada gambar 4 menjelaskan ketika h1 melakukan ping ke h2. Pada gambar tersebut tidak ada kendala ketika h1 ke h2. paket ICMP yang ditunjukkan pada gambar tersebut melewati s1 ke s4. Secara keseluruhan paket yang dilewati yaitu s1-s4-s6-s8-s10-s11.



Gambar 5 Topologi dengan Link ke 1 diputus (DFS)

Berdasarkan Gambar 5 menunjukkan

link ke 1 diputus yaiu link dari s1 ke s4 yang diputus, Sehingga jalur yang dialihkan s1 ke s3, jalur tersebut dapat ditunjukkan ketika paket ICMP melewati s1 ke s3. Secara keseluruhan dari asal ke tujuan yaitu paket melewati s1-s3-s5-s7-s9-s11.

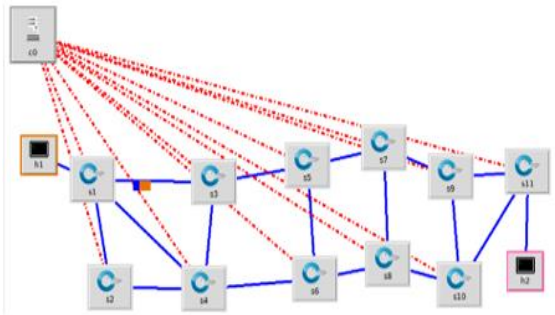


Gambar 6 Topologi dengan Link ke 2 diputus (DFS)

Berdasarkan gambar 6 menunjukkan link ke 2 diputus yaitu link s1 ke s3, sehingga link dialihkan pada alternatif *link* yang lain yaitu s1 ke s2. Berdasarkan penjelasan tersebut dapat dilihat paket ICMP yang melewati s1 ke s2. Secara keseluruhan dalam fungsionalitas pada skenario ini paet melwati beberapa switch yaitu s1-s2-s4-s6-s8-s10-s11.

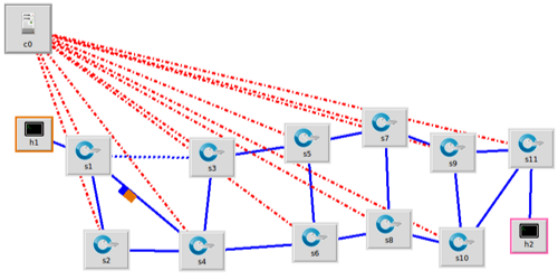
b. Algoritme *Dijkstra* dengan mekanisme *fail path (recovery)*

Pada pengujian ini dijelaskan bagaimana pengujian *fail path (recovery)* dengan algoritme Dijsktra.



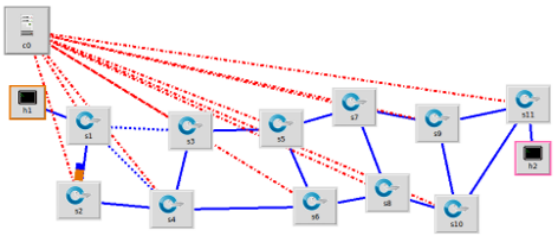
Gambar 7 topologi dengan tidak ada gangguan (Dijkstra)

Berdasarkan pada gambar 6 menunjukkan bahwa paket ICMP melewati s1 ke s3. Dari gambar tersebut belum ada gangguan link pada topologi. Berdasarkan pencarian jalur algoritme Dijkstra Paket tersebut melewati beberapa switch, yaitu s1-s3-s5-s7-s9-s11.



Gambar 8 Topologi dengan Link ke 1 diputus (Dijkstra)

Berdasarkan Gambar 8 menunjukkan link ke 1 diputus yaiu link dari s1 ke s3 yang diputus, Sehingga link yang dialihkan s1 ke s4, jalur tersebut dapat ditunjukkan ketika paket ICMP melewati s1 ke s3. Berdasarkan pada pencarian jalur algoritme Dijkstra switch yang kunjungi paket ICMP yaitu s1-s4-s6-s8-s10-s11.

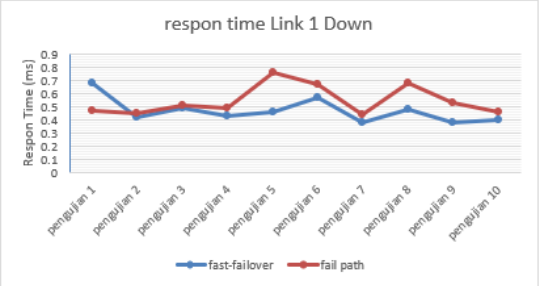


Gambar 9 Topologi dengan Link ke 2 diputus (Dijkstra)

Berdasarkan gambar 9 menunjukkan link ke 2 diputus yaiu link s1 ke s4 yang diputus, Sehingga link yang dialihkan pada *switch* s1 ke s2, jalur tersebut dapat ditunjukkan ketika paket ICMP melewati s1 ke s4. Berdasarkan pada pencarian jalur algoritme *Dijkstra switch* yang kunjungi paket ICMP yaitu s1-s2-s4-s6-s8-s10-s11.

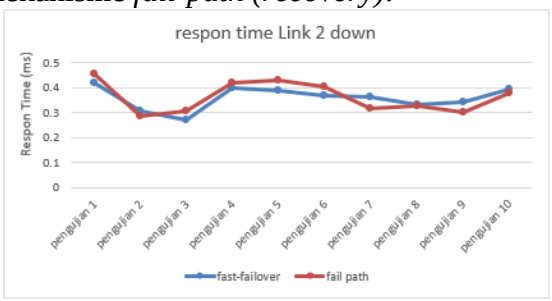
2. Pengujian *Response Time*

Pada pengujian ini dilakukan dengan topologi yang meliputi 11 *switch* dan 2 *host*. *Host* 1 sebagai *client* dan *host* 2 sebagai server. Untuk mengukurt *response time* pada penelitian ini menggunakan tool ping diantara h1 dan h2. Penggunaan ping tersebut dikarenakan terdapat paket ICMP yang digunakan untuk menentukan tujuan dapat dijangkau dan berapa lama paket yang dikirimkan. Maka dari itu pengujian *response time* diambil waktu dari ping yang menentukan waktu ketika jalur dialihkan.



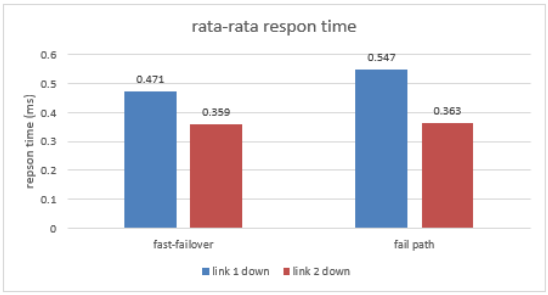
Gambar 10 *Response time* ketika link ke-1 di down

Berdasarkan gambar 10 menjelaskan ketika kedua *link* tersebut diputus atau *link* yang kedua diputus. *Response time* tersebut memiliki beberapa pengujian yang tidak jauh beda. Di grafik diatas lebih unggul DFS dengan Mekanisme *Fast-Failover* daripada mekanisme *fail-path (recovery).*



Gambar 11 *Response time* ketika link ke 2 down

Berdasarkan gambar 11 menjelaskan ketika kedua *link* tersebut diputus atau *link* yang kedua diputus. *Response time* tersebut memiliki beberapa pengujian yang tidak jauh beda. Di grafik diatas lebih unggul mekanisme *fast-failover* daripada mekanisme *fail path (recovery).*

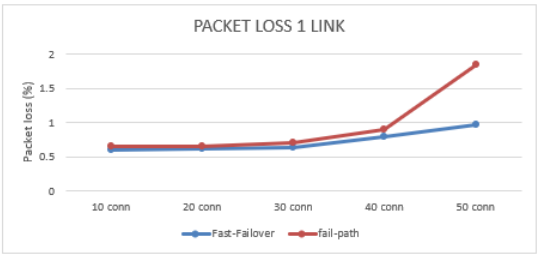


Gambar 12 Rata-rata *response time*

Berdasarkan gambar 12 Dari 2 pengujian *Response time* yang telah dilakukanm dapat diambil kesimpulan bahwa rata-rata dari 2 link yang diputus mekanisme fast-failover memiliki *response time* yang rendah dibandingkan dengan *response time* mekanisme *fail-path (recovery)*

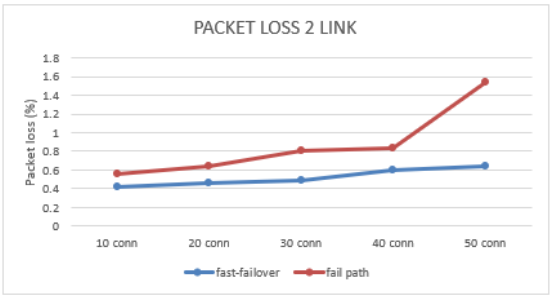
3. Pengujian *Packet Loss*

Berdasarkan hasil pengujian *packet loss* yang telah dilakukan dengan melihat paket yang hilang pada saat pemutusan *link*. Pengujian ini melakukan percobaan dengan 10-50 koneksi dan melakukan 3 link yang diputus pada mekanisme *fast-failover* dan mekanisme *fail path (recovery)*. Dari hasil yang didapatkan, *packet loss* pada pengujian mekanisme *fast-failover* lebih kecil dibandingkan dengan mekanisme *fail path (recovery).*



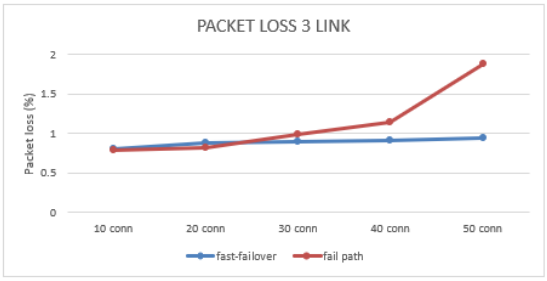
Gambar 13 Grafik *Packet loss* mekanisme *fast-failover* dan *fail path* 1 *link down*

Berdasarkan gambar 13 menjelaskan *packet loss* ketika *link* yang diputus hanya 1 *link*. Berdasarkan dari grafik tersebut *packet loss* mekanisme *fast-failover* lebih rendah dibandingkan dengan mekanisme *fail path (recovery).*



Gambar 14 Grafik *Packet loss* mekanisme *fast-failover* dan *fail path* 2 *link down*

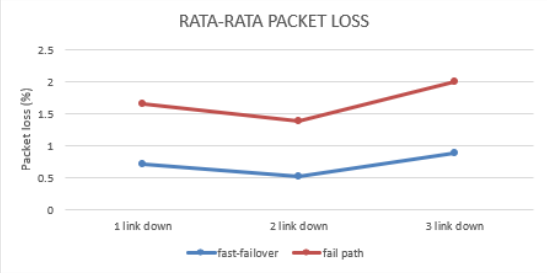
Berdasarkan gambar 14 dijelaskan *packet loss* yang *link* diputus ada 2 *link* yang diputus. Nilai *packet loss* tersebut mekanisme *fast-failover* lebih bagus dari mekanisme *fail path(recovery).*



Gambar 15 Grafik *Packet loss* mekanisme *fast-*

failover dan *fail path* 3 *link down*

Berdasarkan gambar 15 menunjukkan ketika *link* yang diputus ada 3 *link*. hasil *packet loss* tersebut mekanisme *fast-failover* lebih unggul dibandingkan dengan mekanisme *fail-path (recovery)*.



Gambar 16 Rata-rata *Packet loss* mekanisme *fast-failover* dan *fail path*

Berdasarkan gambar 16 Nilai *packet loss* pada kedua mekanisme tersebut tergolong mempunyai *packet loss* yang rendah. Akan tetapi *packet loss* yang di mekanisme *fast-failover* mempunyai nilai yang rendah dibandingkan dengan nilai *fail path (recovery)*. Karena pada saat pemindahan jalur ke jalur yang lain *fast failover* sudah menyediakan *backup link* tanpa komunikasi lagi dengan *controller* yang menyebabkan *packet loss* rendah, sedangkan pada mekanisme *fail path (recovery)* sistem akan berkomunikasi dengan *controller* bila ada *link* yang putus dan *controller* menemukan *link* baru tersebut yang menyebabkan *packet loss* tinggi dibandingkan meknisme *fast-failover*.

4. KESIMPULAN

1. Berdasarkan pengujian fungsionalitas sistem berhasil memantau *bucket* yang berisikan memonitor *outport* pada *OpenFlow Switch*, dan ketika mekanisme *fast-failover* mendeteksi *link* yang *down*, *fast-failover* merespon untuk mengalihkan ke jalur yang lain berdasarkan *bucket* yang dipilih berindikasikan *link up* pada *ouport*nya.
2. Berdasarkan pengujian *packet loss* dan *response time*, Mekanisme *Fast-failover* lebih unnggul daripada mekanisme *fail path (recovery)*, dikarenakan Mekanisme *Fast-failover link* diputus tidak akan berkomunikasi lagi dengan controller sedangkan pada mekanisme *fail path (recovery)* ketika *link* diputus dia akan komunikasi dengan *controller* lagi.

3. DAFTAR PUSTAKA

Dao, S. D. & Marian, R. 2011. Optimisation of precedence-constrained production sequencing and scheduling using genetic algorithms. *Proceedings of the International Multi Conference of Engineers and Computer Scientists*, 16-18 March, Hong Kong.

Antonio Capone, Carmelo Cascone, Alessandro Q.T. Nguyen, & Brunilde Sanso. (2015). Detour Planning For Fast and Reliable Failure Recovery in SDn with Open state. Detour Planning For Fast and Reliable Failure Recovery in SDn with Open state, 1.

Jack Tsai, & Tim Moors. (2006). A review of Multipath Routing Protocols: From Wireless Ad Hoc to Mesh Network. A review of Multipath Routing Protocols: From Wireless Ad Hoc to Mesh Network, 1.

Suyanto. (2014). Artificial Inteligence (2nd ed.). Bandung: Informatika Bandung.

Ying-Dar Lin, Hung-Yi Teng, Chia-Rong Hsu, Chun-Chieh Liao, & Yuan-Cheng Lai. (2016). Fast Failover and Switchover for link Failures and Congestion in Software Defined Network. Fast Failover and Switchover for link Failures and Congestion in Software Defined Network, 1.

Open Networking Foundation. (n.d.). www.opennetworking.org. Retrieved February 23, 2017, from https://www.opennetworking.org/sdn-resources/sdn-definition

Purwiadi, M., Yahya, W., & Basuki, A. (2018). High Availbility Controller Software Defined Network menggunakan Heartbeat dan DBRD.

Rouse, M. (2007). texhtarget.com. Retrieved 2017, from http://searchnetworking.techtarget.com/definition/response-time

RYU SDN Framework Community. (2014). osrg. github.io. Retrieved from http://osrg.github.io/ryu/

Mueller, S., Tsang, R. P., & Ghosal, D. (2003). Multipath Rotuing in Mobile Ad Hoc Network: Issues and Challenge.

pip. (2015). piptools.net. Retrieved 2017, from https://piptools.net/algoritma-dfs-depth-first-search/

Heap, D. (2002). toronto.edu. Retrieved 2017, from http://www.cs.toronto.edu/~heap/270F02/node36.html

Sasmita, W. P., Safriadi, N., & Irwansyah, M. A. (2013). Analisis Quality of Service (QOS) Pada Jaringan Internet (studi Kasus: Fakultas Kedokteran Universitas Tanjungpura).

Rouse, M. (2007). texhtarget.com. Retrieved 2017, from http://searchnetworking.techtarget.com/definition/response-time