Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 2, Februari 2019, hlm. 1648-1654 http://j-ptiik.ub.ac.id

Analisis Kinerja OpenFlow *Switch* Aruba 2920

Zefanya Hermanta[1], Widhi Yahya[2], Achmad Basuki[3]

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

Abstrak

OpenFlow *switch* adalah jaringan *switch* berdasarkan protokol OpenFlow dalam meneruskan paket pada jaringan SDN. Berbagai vendor *switch* sudah menyediakan jaringan OpenFlow-enabled untuk mendukung berbagai versi protokol OpenFlow. Pada implementasi perangkat *switch* terdapat keterbatasan kapasitas dari TCAM pada *line-speed forwarding* dan hanya menampung jumlah aliran yang terbatas. Oleh karena itu, akan menghasilkan kinerja yang berbeda. Dalam penelitian ini, penulis melakukan analisis kinerja OpenFlow *switch* Aruba 2920. Selain itu, menggunakan emulator mininet sebagai referensi pembanding kinerja jaringan. Hasil pengujian menunjukan sistem dapat melakukan pencarian beberapa jalur pada topologi jaringan dan beserta waktu pembentukan jalur atau disebut sebagai *convergance time.* Pengujian menunjukan *throughput* dan *packet loss* pada *multi-path routing* menghasilkan kinerja yang lebih baik, tetapi *convergence time* sedikit lebih lama dari pada *single-path routing*. Kinerja yang dihasilkan pada jaringan OpenFlow *switch* Aruba 2920 sangat rendah jika dibandingkan dengan emulator mininet, karena terdapat keterbatasan *group table* bertipe *select* dan penconcokan *header packet* pada *hardware table* sehingga *flow table* dikelola pada *software table.*

Kata kunci: *multi-path routing, software-defined networking, OpenFlow switch, OpenFlow switch* Aruba 2920

Abstract

OpenFlow switches are network switches based on the OpenFlow protocol in forwarding packets on an SDN network. The various switch vendors have provided OpenFlow-enabled networks to support various versions of the OpenFlow protocol. In the implementation of switch there are capacity limitations of TCAM on line-speed forwarding and only accommodate a limited number of streams. Therefore, it will produce different performance. In this study, the authors conducted an analysis of performance OpenFlow switch Aruba 2920. In addition, it uses the emulator mininet as a reference for comparing network performance. The test results show that the system can search several paths in the network topology and along with the path formation time or referred to as convergance time. Tests show that throughput and packet loss on multi-path routing produce better performance, but convergence time is slightly longer than single-path routing. The resulting performance on the Aruba 2920 OpenFlow switch network is very low compared to the emulator mininet, because there are limitations to the group table type select and thrash packet headers on the hardware table so that the flow table is managed in software table.

Keywords: *multi-path routing, software-defined networking, OpenFlow switch, OpenFlow switch* Aruba 2920

1. PENDAHULUAN

Software-Defined Networking (SDN) telah menentapkan standar baru untuk virtualisasi jaringan dengan memisahkan *data plane* dari *control plane*. *Data plane* diimplementasikan dalam OpenFlow *switch* dan *control plane* sebagai SDN *controller* di implemetasikan pada *software* yang memberi instruksi dalam *forward packet*. Dengan adanya SDN, pemisahan *data plane* dari *control plane* memberikan manfaat dalam meningkatkan kemampuan dalam manajemen trafik. Tidak seperti *traditional network* menggunakan *protokol routing* seperti TCP / IP dan relatif statis pada router untuk menentukan *route packet* dalam suatu jaringan (Pica8 , 2018). OpenFlow *switch* adalah jaringan *switch* berdasarkan protokol OpenFlow dalam meneruskan paket pada jaringan SDN.

OpenFlow switch menggunakan *flow table* dan *group table* untuk melakukan lookups dan *forward packet*. Controller akan bertanggung jawab untuk memberikan logika *routing* dan *forwarding* pada *switch*. OpenFlow *switch* berkomunikasi dengan SDN *controller* menggunakan protokol OpenFlow.

Berbagai vendor *switch* sudah menyediakan jaringan OpenFlow untuk mendukung berbagai versi protokol OpenFlow. Dasar dari protokol OpenFlow setiap *switch* dan *controller* dapat digunakan untuk membagun jaringan akan tetapi itu hanya kemungkinan. Setiap definisi protokol memiliki sudut pandang kasus yang berbeda bagi penulisnya, tidak terkecuali adalah OpenFlow yang membuatnya mungkin dapat diterapkan di *hardware* dan *software system*. Sebagai contoh, OpenFlow 1.0 memiliki sejumlah besar fitur opsional, bahkan yang diperlukan dapat diimplementasikan dengan berbagai cara pada *hardware* yang mendasarinya, dengan hasil yang sangat berbeda (Owens, 2013). Inti dari OpenFlow memiliki 12-tuple dari *field* yang digunakan untuk mencocokan setiap paket yang akan ditangani. Standar tersebut tidak membuat salah satu dari *field* tersebut opsional dan memang semua produksi mendukung OpenFlow *switch*. Dalam penerapan di emulator seperti OpenvSwitch, tidak ada kemungkinan akselerasi *hardware* dari proses pencocokan. Setiap aliran ditangani oleh sistem CPU dan harapannya adalah bahwa kinerja akan ditentukan oleh daya CPU. Pada implementasi dengan perangkat *switch* terdapat keterbatasan kapasitas dari TCAM pada *line-speed forwarding* dan hanya menampung jumlah aliran yang terbatas. Oleh karena itu, akan menghasilkan kinerja yang berbeda jika penerapan dilakukan pada perangkat riil.

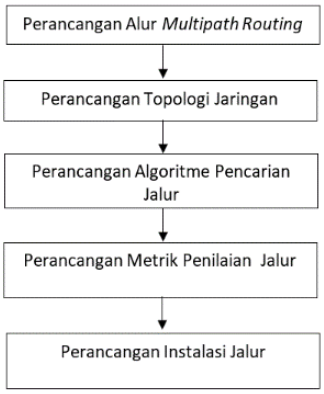
Dalam penelitian ini akan melakukan implementasi protokol *routing* dengan OpenFlow *switch* Aruba 2920. Untuk mengetahui kinerja *protokol routing* pada jaringan OpenFlow *switch* Aruba 2920. Paramater pengujian berupa pengujian pencarian jalur, *throughput*, *packet loss* dan *convergance time*. Selain itu, menggunakan emulator mininet sebagai referensi pembanding kinerja jaringan.

2. KAJIAN KEPUSTAKAAN

Pada penelitian sebelumnya telah melakukan pengembangan jaringan SDN pada OpenFlow *switch*. Penelitiannya berjudul "*Performance Analysis of SDN Switches with Hardware and Software Flow Tables*" (Rygielski, et al., n.d.) dalam penelitiannya telah melakukan analisis kinerja *flow table* pada OpenFlow *switch* Aruba. Namun dalam penelitiannya hanya analisis kinerja setiap *flow table* pada OpenFlow *switch* dan belum menerapkan protokol *routing*. Terdapat beberapa penelitian telah mengimplementasikan protokol *routing* pada jaringan SDN. Penelitian tersebut berjudul "*Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking"* (Jiang, et al., 2014) penelitiannya telah melakukan implementasi *shortest path dengan* algoritme Dijkstra dan Penelitian lain berjudul yang berjudul "*Multi-path Routing* Dengan Load-Balancing pada Open*flow* Software-Defined Network" (Wildan Maulana Syahidillah, 2016) telah melakukan implementasi *multi-path routing* berbasis algoritme DFS. Penelitian-penelitian tersebut masih menggunakan emulator mininet. Oleh karena itu dalam penelitian ini akan melakukan implementasi protokol *routing* dengan OpenFlow *switch* Aruba 2920.

3. PERANCANGAN

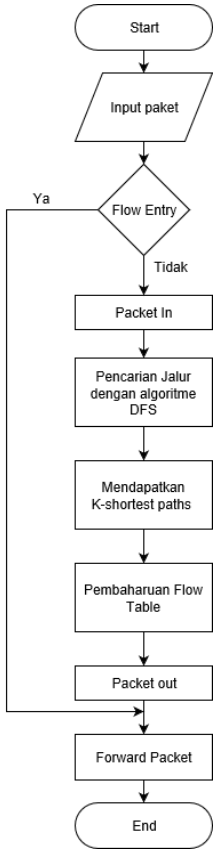
Berikut gambar 1 merupakan langkah-langkah perancangan terhadap lingkungan pengujian yang dibangun berdasarkan kebutuhan yang telah dispesifikasikan.



Gambar 1. Proses Perancangan Siste

3.1. Perancangan Alur *Multi-Path* Routing

Perancangan alur *multi-path routing* menggambarkan bagaimana alur proses dari pengiriman paket dari *source* ke *destination.* Pada gambar 2 merupakan diagram alir *multi-path routing*.

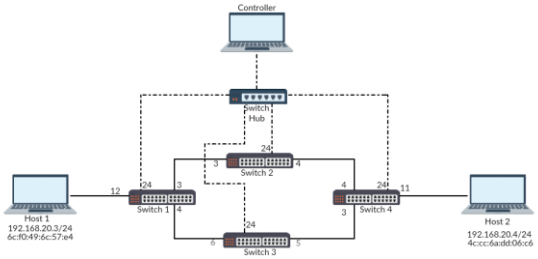


Gambar 2. Diagram Alir *Multi-path* Routing

Pada Gambar 2 merupakan alur *multi-path routing*. Ketika sebuah *switch* menerima sebuah paket yang dikirimkan oleh *host* sumber, maka *switch* akan melakukan pengecekan *header* paket tersebut dengan *flow-entry* yang berada pada *flow table* yang dimiliki oleh perangkat *switch*. Jika ditemukan kesesuaian antara *header* paket dengan *flow-entry*, maka paket akan diteruskan sesuai dengan *action-list* untuk menuju ke *host* tujuan. Jika tidak ditemukan kesesuaian antara *header* paket dengan *flow-entry*, maka *switch* akan melakukan proses enkapsulasi pada paket tersebut kemudian dikirim ke *controller* berupa *PACKET_IN* dengan diberikan tambahan *header* pada paket tersebut seperti *switchID, etherType, etherSrc, etherDst, IPv4src, IPv4dst*. Ketika *PACKET_IN* diterima oleh *controller OpenFlow*, informasi *header* yang dimiliki oleh *PACKET_IN* tersebut akan diperiksa. Setelah *controller OpenFlow* memeriksa informasi dari *PACKET_IN* tersebut, kemudian akan dilakukan pencarian alamat dari *host* sumber dan tujuan dari paket dengan pengetahuannya tentang topologi jaringan *Ethernet* yang terhubung dengannya. Setelah mendapatkan lokasi sumber dan tujuan, kemudian *controller* menjalankan algoritme pencarian jalur dengan algoritme *Depth-First Search* (DFS). Setelah jalur-jalur tersebut ditemukan, kemudian *controller* menyimpan jalur-jalur tersebut berupa daftar beberapa *switch* yang dilalui oleh jalur tersebut. Setelah itu dilakukan seleksi jalur yang telah didapatkan dengan memilih *cost* terendah sesuai jumlah jalur yang telah ditentukan. Setelah itu dilakukan pembaharuan *flow table* pada switch, kemudian paket tersebut akan diproses berdasarkan *instructions. Kemudian PACKET_OUT digunakan controller* mengirimkan paket keluar dari *port* tertentu pada *switch* dan paket diteruskan (*forward packet*) menuju *destination.*

3.2. Perancangan Topologi

Pada perancangan topologi untuk memanfaatkan *multi-path routing* pada suatu jaringan, diperlukan suatu topologi yang memiliki berbagai jalur yang redundan yang dapat dilalui suatu node jaringan. Jalur redudan merupakan jalur alternatif yang digunakan untuk meningkatkan ketersedian jaringan, sehingga memiliki beberapa jalur dari sumber ke tujuan. Oleh karena itu, penelitiaan ini menggunakan topologi dengan jalur redundan untuk menilai keberhasilan *multi-path routing* yang telah diterapkan. Berikut ini topologi yang dapat digunakan untuk untuk *multi-path routing* pada gambar 3.



Gambar 3 Topologi Jaringan

Pada gambar 3 topologi jaringan tersebut menggunakan mekanisme *out-of-band* dimana setiap port pada *switch* dihubungkan ke satu *controller*. Untuk menghubungkan ke empat *switch* ke *controller* menggunakan *switch hub.* Pada proses perancangan topologi terdapat aturan konfigurasi *OpenFlow* pada arsitektur *OpenFlow switch* Aruba 2920. Penulis menggunakan *mode virtualization* dimana mode ini memiliki aturan memisahkan antara *OpenFlow controller* dengan *OpenFlow switch* dengan VLAN, agar *traffic* pada *control plane* tidak melintasi *data plane.*

Berikut pengalamatan IP pada setiap perangkat:

- Host 1: 192.168.20.3/24 dengan *MAC address*: 6c:f0:49:6c:57:e4

- Host 2: 192.168.20.4/24 dengan MAC *address*: 4c:cc:6a:dd:06:c6
- *Switch* 1: 192.168.10.33/24
- *Switch* 2: 192.168.10.30/24
- *Switch* 3: 192.168.10.34/24
- *Switch* 4: 192.168.10.30/24
- *Controller*: 192.168.10.31/24

Berikut manajeman VLAN dalam topologi tersebut:

- VLAN 10 merupakan vlan *Controller*
- VLAN 20 merupakan vlan Open*Flow*

Berikut Datapath-ID setiap *switch*:

- *Switch* 1: 5752715434613568
- *Switch* 2: 5792332529910336
- *Switch* 3: 5792332529885568
- *Switch* 4: 5792332517563328

3.3. Perancangan Algoritme Pencarian Jalur

Algoritme pencarian jalur menggunakan algoritme DFS. Algortima DFS mempunyai metode pencarian jalur dengan menggunakan struktur data stack yang berfungsi untuk menyimpan rute, *backtracking*, dan dapat melakukan ekspansi mecari seluruh jalur untuk dilewati untuk menuju ke node yang telah ditentukan.

Pseudocode: Algoritme DFS Modifikasi	
1	**DFS**(G, s, d)
2	P array to store paths
3	stack initialiased as (s, [s])
4	**while** stack is not empty
5	pop stack, save to node, p
6	**for** next in (G[node] – p)
7	**if** next is d
8	append p + [next] to P
9	**else**
10	push (next, p + [next]) to stack
	end for
11	**end while**
12	**return** P
13	**end**
14	

Pada *pseudocode* DFS Modifikasi tersebut merupakan pencarian jalur. G menyatakan sebuah *adjacency matrix* dari suatu grafik, yaitu matriks yang menyimpan informasi ketetanggaan dari suatu node pada suatu topologi. *Pseudocode* tersebut akan menghasilkan seluruh jalur yang dapat dilalui dari node *source* ke tujuan *destination* yang disimpan pada sebuah *array list P*.

3.4. Perancangan Metrik Penilaian Jalur

Metrik penilaian dari suatu jalur untuk memberi nilai suatu jalur mengunakan kalkulasi metrik penilaian jalur atau *link cost* pada protokol OSPF seperti pada persamaan (1) (Cisco 2005).

$$OSPF\ Cost = \frac{B_R}{B_L} \quad (1)$$

Berdasarkan Persamaan (1), B_R menyatakan *reference bandwidth* yang digunakan protokol OSPF, yaitu sebesar 1 Gbps, sedangkan B_L menyatakan *link bandwidth* dari sepasang *router* yang berhubungan 1 Gbps. Berdasarkan hasil kalkulasi tersebut didapat nilai dari bobot sebuah jalur (*path weight*) pada semua jalur sama dengan 1 seperti pada persamaan (2)

$$pw\ (p) = 1 \quad (2)$$

Setelah mendapatkan *path cost* antara *switch-switch* dilakukan penjumlahan seluruh jalur dari *source* menuju *destination* yang di dapat dari fungsi pencarian jalur. Kemudian jalur-jalur tersebut dipilih beberapa jalur terbaik berdasarkan nilai dari bobot sebuah jalur (*path weight*) sebanyak jumlah jalur yang telah ditentukan dan jalur-jalur yang terpilih digunakan untuk proses *routing.*

3.5. Perancangan Instalasi Jalur

Proses instalasi jalur menggunakan metode *multi-path*. Metode *multi-path* merupakan bagian dari *group table* bertipe *select*. *Group table* bertipe *select* dalam algoritme pemilihan jalur dapat mengimplementasikan *equal load sharing* dan juga dapat optional berdasarkan *bucket weight*. Pada penelitian ini menggunakan *bucket weight* dalam pemilihan jalur, tetapi dalam penerapannya *switch* Aruba 2920 mengharuskan *group table* bertipe *select* yang menggunakan *bucket weight* harus melakukan *set* pada *weight* sama dengan 1 karena pada *switch* Aruba 2920 tidak ada mekanisme *weight* (Hewlett Packard Enterprise Development LP, 2017). Jadi pada instalasi jalur akan menghasilkan pembagian beban yang relatif seimbang. Untuk mengatur *weight* sama dengan 1 dengan cara menetapkan *bucket weight* sama dengan 1 yang terdapat pada *action bucket.* Untuk itu perhitungan persentase bobot dibuat seimbang dari masing-masing jalur, seperti pada persamaan 3.

$$w(p) = 1 \quad (3)$$

4. IMPLEMENTASI

Implementasi dilakukan pada jaringan riil menggunakan 2 *host*, 4 *switch* Aruba 2920, 1 *switch hub* dan 1 *controller*. *Switch* Aruba 2920 menggunakan *firmware version* 16.04. *Flow table* dikelola pada *table software*. Topologi jaringan dibuat dengan mekanisme *out-of-band*. Arsitektur OpenFlow pada *switch* Aruba 2920 menggunakan mode *virtualization*. Pengembangan program *controller* menggunakan Ryu dengan Bahasa pemrograman Python. Gambaran umum dari implementasi jaringan SDN dengan *switch* Aruba 2920 dapat dilihat pada gambar 4.

Gambar 4. Implementasi Jaringan SDN dengan *switch* Aruba 2920

5. HASIL PENGUJIAN DAN PEMBAHASAN

5.1. Pengujian Pencarian Jalur

Pada pengujian pencarian jalur akan membandingkan antara *single-path routing* berbasis algoritme dijkstra dengan *multi-path routing* berbasis algoritme DFS. Untuk mendapatkan jalur maka dilakukan pengujian dengan cara mengirimkan paket ICMP menggunakan aplikasi *ping* dari sumber host 1 ke tujuan host 2.

Tabel 1. Hasil Pengujian Pencarian Jalur

	Datapath-ID	**Keterangan**
Single-path	[5752715434613568, 5792332529885568, 5792332517563328]	[*switch* 1, *switch* 3 dan *switch* 4]
Multi-path	1. [5752715434613568, 5792332529910336, 5792332517563328] 2. [5752715434613568, 5792332529885568, 5792332517563328]	1. [*switch* 1, *switch* 3 dan *switch* 4] 2. [*switch* 1, *switch* 2 dan *switch* 4]

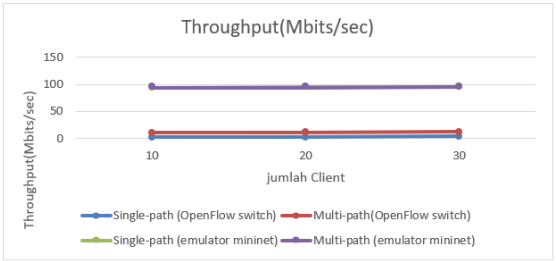
Pada tabel 1 merupakan hasil pengujian pencarian jalur *single-path routing* berbasis algoritme Dijkstra dan pencarian jalur *multi-path routing* berbasis algoritme DFS. Pada pengujian ini single path terbukti menghasilkan satu jalur dan pada *multi-path* menghasilkan dua jalur

5.2. Pengujian *Throughput*

Pengujian *troughput* dijadikan acuan dalam menetukan kinerja/*performance* koneksi jaringan. Pada scenario pengujian *throughput* membandingkan *performance* antara *single-path routing* berbasis algoritme Dijkstra dengan *multi-path routing* berbasis algoritme DFS dalam pengiriman data. Untuk mengukur *throughput*, menggunakan aplikasi iperf. Pada pengujian ini, *host* 1 sebagai client mengirimkan trafik TCP selama 10 detik dengan menggunakan jumlah koneksi *client* sebanyak 10, 20 dan 30 ke *host* 2 sebagai server. Berikut ini hasil dari pengujian *throughput* pada table 3.

Tabel 2. Hasil Pengujian *Throughput*

Jumlah Client	***Single-path (Mbits/sec)***		***Multi-path (Mbits/sec)***	
	Emulator Mininet	OpenFlow switch Aruba	Emulator Mininet	OpenFlow switch Aruba
10	94,9	1,93	95,7	10,7
20	95,1	2,23	95,7	11,4
30	95,1	3,86	95,8	11,6
Rata-rata	95,03	2,67	95,73	11,24



Gambar 5. Grafik Hasil Pengujian *Throughput* Single-path dan *Multi-path* pada OpenFlow *switch* Aruba dan emulator mininet

Berdasarkan hasil pengujian *throughput* pada tabel 2 dan gambar 5, *single-path routing* dan *multi-path routing* terjadi peningkatan *throughput* pada setiap penambahan jumlah *client* 10 ,20 dan 30. Jika diambil rata-rata *throughput* pada *single-path 2,67 Mbit/s dan pada multi-path 11,24 Mbit/s.* Dengan demikian kinerja *multi-path routing* lebih baik dari pada *single path routing.* Pada pengujian ini terbukti penggunaan *multi-path routing* mendistribusikan *traffic* ke beberapa jalur dan menghasilkan kinerja yang lebih baik dari pada penggunaan *single-path routing* yang hanya memanfaatkan penggunaan satu jalur saja.

Pada pengujian *throughput* pada *single-path routing* terlihat kinerja yang dihasilkan sangat kecil yaitu dengan rata-rata 2,67 Mbits/sec dan penggunaan *multi-path routing* dengan rata-rata 11,24 Mbits/sec dinilai belum maksimal. Jika dibandingkan dengan emulator mininet, *throughput* yang dihasilkan lebih baik menggunakan emulator mininet. Hal ini disebabkan karena adanya keterbatasan pada *switch* Aruba 2920 dimana pada *multi-path routing* menggunakan tipe *group select* yang hanya bisa menggunakan *software table* dan pada *single-path routing* pada proses *match*ing header pada header ETH_DST tidak terdapat di *hardware table* sehingga diahlikan ke *software table*. Pada *software table match* rules disimpan pada SDRAM menyebabkan kinerja lebih kecil, tetapi jika menggunakan *hardware table match* rules akan disimpan pada TCAM (*ternary content-address-able memory*) akan menghasilkan kinerja lebih baik. Pada penelitian yang dilakukan oleh Piotr Rygielski, Marian Seliuchenko, Samuel Kounev dan Mykhailo Klymash tentang "*Performance Analysis of SDN Switches with Hardware and Software Flow Tables*" mereka melakukan pengujian dengan membandingkan kinerja antara *software table* dan *hardware table* terbukti pada pengujian mereka dengan menggunakan *software table* menghasilkan kinerja lebih kecil.

5.3. Pengujian *Packet Loss*

Pengujian *packet loss* dilakukan untuk mengetahui persentasi kegagalan dalam mengirimkan suatu packet dari *client* ke *server*. Pada pengujian ini dilakukan perbandigan packet loss antara *single-path routing* berbasis algoritme Dijkstra dengan *multi-path routing* berbasis DFS. Dalam pengujian host 1 sebagai *client* dan host 2 sebagai *server*. Pada sisi *client* mengirimkan paket UDP dengan penggunakan aplikasi iperf. Berikut pada table 4 merupakan hasil dari pengujian *packet loss.*

Tabel 3. Hasil Pengujian *Packet Loss*

Jumlah Client	Rata-Rata Packet Loss (%)			
	Single-path (Mbits/sec)		***Multi-path (Mbits/sec)***	
	Emulator Mininet	OpenFlow switch Aruba 2920	Emulator Mininet	OpenFlow switch Aruba 2920
10	0,44 %	51 %	0,11 %	5,2 %
20	0,99 %	75 %	0,11 %	63 %
30	1,11 %	90 %	0,44 %	65 %
Rata-rata	0,84 %	72 %	0,22 %	44,4 %

Pada tabel 3 merupakan hasil pengujian *packet loss* pada *single-path* dan *multi-path*. Hasil pengujian rata-rata *packet loss single-path routing* memiliki persentasi 72% dan *multi-path routing* memiliki persentasi 44,4%. Dari data tersebut bisa tergambar bahwa *packet loss* pada *single-path routing* lebih tinggi dari pada *multi-path routing*. Perbedaan ini disebabkan karena *single-path routing* hanya menggunakan satu jalur dan tidak memiliki jalur untuk berbagi trafik dalam mencapai tujuan, sehingga terjadi *bottleneck link* pada sebuah jalur sedangkan pada *multi-path routing* memanfaatkan beberapa jalur dalam mendistribusikan atau berbagi trafik ke beberapa jalur yang terdapat pada topologi jaringan.

5.4. Pengujian *Convergence Time*

Pengujian *convergence time* adalah untuk mengetahui waktu pemprosesan dari suatu algoritme *routing*. Dalam pengujian *convergence time* dilakukan pengiriman paket ICMP menggunakan aplikasi ping dari *host* 1 sebagai *source* ke host 2 sebagai destination. Berikut table 5 merupakan hasil dari pengujian *convergance time*.

Tabel 4. Hasil Pengujian *Convergence Time*

***Convergence Time* (*ms*)**			
Single-path		***Multi-path***	
Emulator Mininet	OpenFlow switch Aruba 2920	Emulator Mininet	OpenFlow switch Aruba 2920
0.00104	0.00101	0.00231	0.00225

Pada Tabel 4 merupakan hasil pengujian *convergence time* yang didapat berdasarkan waktu yang dibutuhkan dalam pembentukan seluruh rute yang berada pada *data plane*. Hasil yang didapat menujukan *single path routing* lebih cepat dibandingkan dengan *multi-path routing*. Hal ini terjadi karena *controller* pada *multi-path routing* memerlukan waktu pemprosesan yang lebih lama dibandingkan *single path routing*

6. KESIMPULAN DAN SARAN

Bedasarkan hasil pengujian kinerja *single-path routing* pada OpenFlow *switch* Aruba2920 didapat hasil rata-rata *throughput* 2,67 Mbit/sec, *packet loss* 72 % dan *convergence time 0.00101 ms dan* pada emulator mininet didapat hasil rata-

rata *throughput* 95,03 Mbit/sec, *packet loss* 0,84% dan *convergence time* 0.00104 *ms.* Pengujian *multi-path routing* pada OpenFlow *switch* Aruba didapat hasil rata-rata *throughput* 11,24 Mbit/sec, *packet loss* 44,4% dan *convergence time 0.001225 ms* dan pada emulator mininet didapat hasil rata-rata *throughput* 95,73Mbit/sec, *packet loss* 0,22% dan *convergence time* 0.00231 *ms.* Berdasarkan hasil rata-rata *throughput* dan *packet loss* pada *multi-path routing* kinerja yang dihasilkan lebih baik, tetapi dalam pembentukan jalur/*convergence time* lebih lambat dari pada *single-path routing*. Jika dibandingan dengan emulator mininet kinerja yang dihasilkan lebih baik dari pada OpenFlow *switch* Aruba 2920. Hal tersebut disebabkan karena *flow table* dikelola pada *software table* sehingga kinerja pada OpenFlow *switch* Aruba tidak maksimal.

Pada penelitian ini *flow table* dikelola di *software table* karena metode *multi-path* menggunakan *group table* bertipe *select* untuk penelitian selanjutnya bisa menggunakan *group table* bertipe *all* untuk mendukung penggunaan *hardware table.*

7. DAFTAR PUSTAKA

Banner, Ron, and Ariel Orda. 2005. "Multipath Routing Algorithm for Congestion Minization."

Cisco. 2005. *OSPF Design Guide.* August 10. http://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html.

Heap, Danny. 2002. *Depth-first search (DFS).* Accessed April 20, 2018. www.cs.toronto.edu/~heap/270F02/node36.html.

Hewlett Packard Enterprise Development LP. 2017. *ArubaOS-Switch OpenFlow v1.3 Administrator Guide for 16.04.*

Jiang, J.R., H.W. Huang, J.H. Liao, and S.Y. Chen. 2014. *Extending Dijkstra's Shortest Path Algorithm for Software.* IEEE.

Kreutz, Diego, Paulo Esteves Verissimo, Siamak Azodolmolky, Fernando M. V. Ramos, Christian Esteve Rothenberg, and Steve Uhlig. 2015. "Software-Defined Networking: A Comprehensive Survey." *Proceedings of the IEEE.* IEEE.

Lin, Chuan-Bi, Te-Chu H, and Hsiang-Yu Chan. 2017. "The Implementation of Multi-Path Delivery for Data Flows Using Raspberry Pi Boards in Software-Defined Networks."

Nunes Astuto, Bruno, Marc Mendonça, Xuan Nam Nguyen, Katia Nam Nguyen, and Thierry Turletti. 2014. *A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks.* IEEE Communications Society.

Open Networking Foundation. 2012. "Software-Defined Networking: The New Norm for Networks." Bayshore Road.

Owens, Bill. 2013. *OpenFlow Switching Performance: Not All TCAM Is Created Equal.* Accessed Desember 23, 2018. https://packetpushers.net/openflow-switching-performance-not-all-tcam-is-created-equal/.

Pica8 . 2018. *OpenFlow switches* . Accessed desember 20, 2018. https://www.pica8.com/openflow-switch/.

Rygielski, Piotr, Marian Seliuchenko, Samuel Kounev, and Mykhailo Klymash. n.d. *Performance Analysis of SDN Switches with Hardware and Software Flow Tables.* German Research Foundation (DFG).

Wildan Maulana Syahidillah, Widhi Yahya, Sabriansyah Rizqika A. 2016. "Multipath Routing Dengan Load-Balancing Pada Openflow Software-Defined Network." Malang.

Yadav, Akash, Ashok Singh Sairam, and Arka Prokash Mazumar. 2017. *Selecting Stable Route in Multipath Routing Protocols.* Patna: Indian Institute of Technology Patna.