Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 1, No. 12, Desember 2017, hlm. 1842-1849 http://j-ptiik.ub.ac.id

Analisis Perbandingan Algoritma *Floyd-Warshall* dan *Dijkstra* untuk Menentukan Jalur Terpendek Pada Jaringan *Openflow*

Ibrahim Attamimi[1], Widhi Yahya[2], Mochammad Hannats Hanafi[3]

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

Abstrak

Software Defined Network (*SDN*) merupakan suatu pemodelan jaringan yang memisahkan antara *control plane* dan *data plane* dengan menggunakan standar komunikasi *protocol Openflow* untuk menghubungkan keduanya. *SDN* mulai dikembangkan beberapa tahun terakhir ini dan sudah banyak diimplementasikan salah satunya adalah untuk *routing* jaringan. Terdapat dua algoritma *routing* dalam penelitian ini yaitu *Dijkstra* dan *Floyd-Warshall*. Kedua algoritma ini akan diimplementasikan menggunakan *emulator Mininet* dan *Ryu controller*. Pengujian yang dilakukan meliputi parameter *convergence time*, *throughput*, *link failure, resource usage*. Pada pengujian *convergence time*, *Dijkstra* lebih unggul dengan rata-rata sekitar 9,08-16,03ms dibandingkan dengan *Floyd-Warshall* dengan rata-rata sekitar 14,51-38,63ms. Berdasarkan hasil pengujian throughput, kedua algoritma tidak terlalu memiliki perbedaan yang signifikan. Pada *Floyd-Warshall* memiliki rata-rata *throughput* sekitar 12,7-29,54 Gbps sedangkan *Dijkstra* memiliki rata-rata sekitar 12,89-29,22 Gbps. Untuk hasil pengujian dari *link failure*, *Floyd-Warshall* lebih unggul karena memiliki *recovery time* lebih cepat dengan rata-rata sekitar 30,34-38,51 detik dibandingkan dengan *Dijkstra* dengan rata-rata 37,65-44,92 detik. Untuk hasil dari pengujian *resource usage*, pada pengukuran *memory usage* kedua algoritma sama-sama mengkonsumsi *memory* sama besar namun pada pengukuran *CPU Usage*, *Floyd-Warshall* mengkonsumsi daya *CPU* lebih besar dibandingkan dengan *Dijkstra*.

Kata kunci: *SDN, OpenFlow, Mininet, Ryu, Dijkstra, Floyd-Warshall*

Abstract

Software Defined Network (SDN) is network modeling that separates between control plane and data plane using Openflow protocol communication standard to connect both of them. SDN began to developed in recent years and has been widely implemented, one of them is network routing. There are two routing algorithms in this research which are Dijkstra and Floyd-Warshall. Both of these algorithms will be implemented using Mininet emulator and Ryu controller. Tests that performed in this research include parameters such convergence time, throughput, link failure, and resource usage. The result of convergence time test from this research showed Dijkstra's is superior to average of about 9,08-16,03milliseconds compared with the Floyd-Warshall with average of about 14,51-38,63milliseconds. Based on the results of throughput testing, both algorithms do not have significant differences. The Floyd-Warshall had an average throughput of 12.7-29.54Gbps while Dijkstra's had an average of about 12.89-29.22Gbps. For the test results from the Link failure, Floyd-Warshall is superior because it had a faster recovery time with average about 30.34-38.51seconds compared to Dijkstra's with average of 37.65-44.92seconds. Usage resource test showed that memory usage measurements for both algorithms share same memory, but CPU usage measurements showed the Floyd-Warshall consumes more CPU power compared to Dijkstra's.

Keywords: *SDN, OpenFlow, Mininet, Ryu, Dijkstra, Floyd-Warshall*

1. PENDAHULUAN

Teknologi komputer yang telah berkembang pesat telah membawa berbagai macam perubahan. Salah satunya dalam hal jaringan, kini jaringan statis konvensional sudah mulai digantikan dengan jaringan dinamis yang bersifat *programmable*. *Software Defined Network* (*SDN*) merupakan salah satu contoh dari perkembangan suatu jaringan. Jaringan

SDN memiliki konsep yang berbeda dengan jaringan tradisional, dimana pada jaringan tradisional *control plane* digabung dengan *data plane* pada satu perangkat, hal tersebut berbeda dengan konsep yang ditawarkan pada jaringan *SDN* dimana *control plane* dipisahkan dari *data plane*. Pemisahan yang dimaksud adalah pemisahan *router* atau *switch* yang ada pada *data plane* yang biasanya berfungsi untuk *forwarding* paket data dan memberikan tanggung jawab kepada *software* tertentu pada *control plane* sebagai *controller* yang secara logis terpusat untuk mengontrol perilaku sebuah jaringan (Astuto, 2014).

Tujuan utama dari *SDN* sendiri adalah untuk mempermudah penerapan berbagai jenis macam aplikasi jaringan yang dapat diprogram pada *controller*-nya, karena sifat *controller SDN* yang *programmable* tersebut. Macam-macam contoh aplikasi jaringan yang dapat diterapkan pada *SDN* seperti *Load balancing, Network Virtualization, Intrusion Detection, Routing,* dan lain-lain (McKeown, 2008). Berkaitan dengan *Routing*, karena perkembangan jaringan komputer yang semakin pesat ini menimbulkan masalah dalam jaringan komputer yang sering kali antara satu komputer dengan komputer lain terpisah jauh secara geografis, Hal ini menimbulkan kebutuhan yang tinggi terhadap performansi jaringan dan kontrol jaringan yang semakin rumit dan tidak fleksibel. Kinerja suatu jaringan dapat dipengaruhi oleh beberapa faktor seperti konfigurasi jaringan, jumlah permintaan, dan metode manajemen jaringan tersebut. Salah satu yang faktor yang mempengaruhi manajemen jaringan tersebut adalah network routing. Routing merupakan penentuan jalur terbaik yang akan dilalui informasi yang dikirim dari pengirim menuju penerima. Dalam *network routing* biasanya digunakan suatu algoritma pencarian jalur terpendek atau disebut *Shortest Path Problem Algorithm* (Saputra, 2016). Terdapat beberapa macam algoritma *Shortest Path*, salah satunya adalah Algoritma *Floyd-Warshall* dan *Dijkstra*. Kedua algoritma tersebut memiliki cara pencarian yang berbeda, pada Algoritma *Floyd-Warshall* menggunakan pencarian jalur all pair shortest path yang artinya pencarian jalur terpendek dapat ditentukan antara semua pasangan simpul, sedangkan pada Algoritma *Dijkstra* yang menggunakan pencarian jalur single source shortest path yang artinya pencarian jalur terpendek dapat ditentukan dari simpul tertentu ke semua simpul yang lain (Jiang, 2014).

Dalam penelitian ini akan dimplementasikan algoritma *Floyd-Warshall* dan *Dijkstra* sebagai penentuan jalur yang akan dipilih paket informasi yang dikirimkan dari sumber ke tujuan menggunakan *controller Ryu* dan memakai *module NetworkX* pada beberapa macam topologi pada *emulator Mininet*., Dengan melakukan analisis melihat dari *scalability, convergence time, throughput, recovery time,* dan *resource usage.* untuk menentukan algoritma manakah yang memiliki performa labih bagus dan lebih baik untuk digunakan dalam menentukan jalur terpendek yang harus dilalui oleh paket dari pengirim menuju penerima pada jaringan *Openflow.*

2. DASAR TEORI

1. *Software Defined Network*

Metode jaringan *Software-Defined Network* adalah sebuah metode dalam bebuat suatu jaringan komputer dengan memisahkan jalur data dan jalur kontrol, dimana pada metode ini switch akan berfungsi sebagai jalur data sedangkan untuk jalur kontrol terdapat pada kontroler (Lantz, 2010). Tujuan dari metode jaringan *SDN* ini diantaranya adalah untuk memudahkan dalam membangun suatu jaringan komputer, memudahkan dalam memelihara. Prinsip kerja dari *switch* pada metode jaringan *SDN* antara lain adalah menghubungkan antara *switch* dengan *switch*, switch dengan kontroler, dan *switch* dengan *host*. Dalam metode jaringan *SDN* terdapat salah satu protokol yang digunakan untuk menghubungkan antara kontroler dengan switch yaitu protokol *OpenFlow*, pada *protocol OpenFlow* terdapat *OpenFlow switch* yang berfungsi sebagai *switch* dan *OpenFlow* kontroler yang berfungsi sebagai kontroler. *OpenFlow* adalah standar pertama mengenai antarmuka komunikasi antara *control layer* dan *forwarding layer* dalam arsitektur *SDN*.

2. OpenFlow

OpenFlow merupakan sebuah standar protokol komunikasi terbuka antara *data plane* dan *control plane* pada *SDN*. Berbeda dengan jaringan konvensional, dimana fungsi *data plane* dan *control plane* berada pada satu perangkat *switch*/*router*, jaringan yang dipasang *OpenFlow* memungkinkan fungsi *data plane* dan *control plane* berada pada perangkat yang terpisah, sehingga memudahkan melakukan ektensi atau penambahan fungsi pada suatu

jaringan. Switch pada *OpenFlow* hanya merupakan perangkat *data plane* yang hanya melakukan fungsi penerusan paket berdasarkan suatu set kriteria yang menyeleksi paket dan suatu set tindakan atau action yang diterapkan pada paket tersebut. Satuan dari set kriteria beserta tindakannya adalah yang disebut sebagai flow. Sedangkan perangkat *control plane* pada *OpenFlow* dapat berupa server yang menyediakan logika dan kecerdasan dari jaringan atau yang disebut sebagai *Controller* atau *Network Operating Sistem* (NOS) (Dely, 2011).

C. *Mininet*

Mininet adalah sebuah *emulator* jaringan yang membangun jaringan yang terdiri atas *host virtual*, *switch*, *controller* dan *link*. *Mininet* merupakan sebuah sistem untuk melakukan rapid prototyping jaringan yang sangat besar pada sumber daya yang terbatas, seperti sebuah laptop (Saputra, 2016). *Mininet* menggunakan virtualisasi berbasis proses untuk menjalankan banyak *host* dan *switch* pada satu *kernel* OS. *Mininet* memanfaatkan *virtual software switch OpenvSwitch* untuk membangun topologi *OpenFlow SDN*. *Mininet* di sini akan digunakan untuk simulasi jaringan *SDN* dengan berbagai macam topologi.

D. *Ryu*

Ryu merupakan sebuah *framework* berbasis komponen untuk *SDN*. *Ryu* menyediakan komponen *software* dengan API yang memudahkan *developer* untuk membangun aplikasi-aplikasi kontrol dan manajemen jaringan (Saputra, 2016). Pengembangan aplikasi untuk *Ryu* dapat dilakukan dengan menggunakan bahasa *Python*. *Ryu* mendukung berbagai protokol untuk memanajemen jaringan antara lain *OpenFlow, NetConf*, Of-*config* dll. Kebutuhan atas *Ryu* sebagai *OpenFlow controller* adalah karena *Ryu* mendukung *OpenFlow* versi 1.0 hingga 1.5.

E. Algoritma *Dijkstra*

Algoritma *Dijkstra* merupakan salah satu jenis dari algoritma greedy. Algoritma *greedy* adalah algoritma pemecahan persoalan yang berhubungan dengan masalah optimasi yang cukup terkenal. Sesuai dengan artinya sendiri greedy, yaitu rakus atau tamak namun bukan dalam konteks yang negatif, algoritma ini hanya memikirkan solusi apa yang terbaik untuk dilakukan pada setiap langkanya tanpa memikirkan konsekuensi ke depannya dan keputusan yang sudah diambil tidak dapat diubah kembali. Jadi dapat dikatakan algoritma greddy ini berusaha agar seiap langkah yang dilakukukannya memiliki nilai yang optimum lokal, berharap agar nilai optimum lokal yang didapatkannya ini mengarah kepada nilai optimum global (Noviandi, 2007).

```
1.  function Dijkstra(Graph, source):
2.      for each vertex v in Graph:
3.      dist[v] := infinity ;
4.          previous[v] := undefined ;
5.      end for
6.      dist[source] := 0 ;
7.      Q := the set of all nodes in Graph
8.      while Q is not empty:
9.          u := vertex in Q with smallest distance in dist[] ;
10.         remove u from Q ;
11.         if dist[u] = infinity:
12.             break ;
13.         end if
14.         for each neighbor v of u:
15.             alt := dist[u] + dist_between(u, v) ;
16.             if alt < dist[v]:
17.                 dist[v] := alt ;
18.                 previous[v] := u ;
19.                 decrease-key v in Q;
20.             end if
21.         end for
22.     end while
23.  return dist;
```

Gambar 1 *Pseudocode* Algoritma *Dijkstra*

E. Algoritma *Floyd-Warshall*

Jika Algoritma *Dijkstra* sebelumnya merupakan salah satu dari jenis Algoritma Greedy, Algorima *Floyd-Warshall* merupakan salah satu jenis dari pemrograman dinamis. Pemrograman dinamis merupakan suatu cara memecahkan suatu masalah dengan cara memandang solusi yang akan didapatkan sebagai keputusan yang saling terkait. Solusi yang dibentuk ini didapatkan dari solusi-solusi yang berasal dari tahap sebelumnya dan memiliki kemungkinan bahwa solusi yang didapatkan lebih dari satu (Noviandi, 2007).

Yang membedakan dengan algoritma *Greedy* adalah jika pada algoritma *Greedy* pada setiap langkah yang dilakukannya hanya berdasarkan nilai optimum yang didapatkan tanpa memikirkan konsekuensi kedepannya jika memilih suatu langkah pada suatu tahap. Karena hal tersebutlah algoritma *greedy* gagal memberikan solusi yang terbaik. Berbeda dengan pemrograman dinamis yang mencoba memberikan solusi dengan memikirkan konsekuensi yang mungkin akan terjadi kedepannya jika mengambil suatu langkah. Prinsip optimalitas merupakan prinsip yang

dipakai oleh pemrograman dinamis, jika solusi total optimal, maka bagian dari solusi sampai pada suatu tahap (misalnya tahap ke-i) juga optimal (Noviandi, 2007).

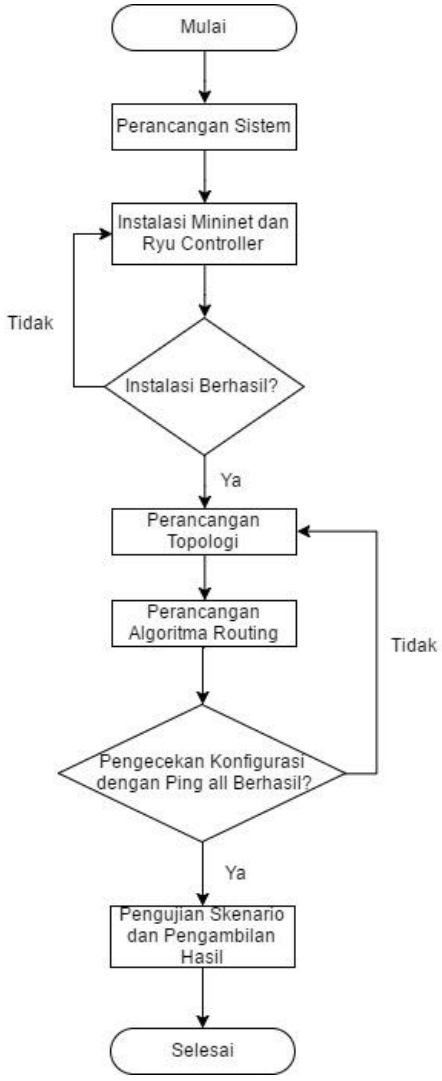
```
1.  function fw(int[1..n,1..n] graph)
2.  {
3.  // Inisialisasi
4.  var int[1..n,1..n] jarak := graph
5.  var int[1..n,1..n] sebelum
6.  for i from 1 to n
7.  for j from 1 to n
8.  if jarak[i,j] < Tak-hingga
9.     sebelum[i,j] := i
10.
11. // Perulangan utama pada algoritma
12. for k from 1 to n
13. for i from 1 to n
14. for j from 1 to n
15. if jarak[i,j] > jarak[i,k] + jarak[k,j]
16.    jarak[i,j] = jarak[i,k] + jarak[k,j] sebelum[i,j] = sebelum[k,j]
17. return jarak
18. }
```

Gambar 2 *Pseudocode* Algoritma *Floyd-Warshall*

3. PERANCANGAN DAN ANALISIS

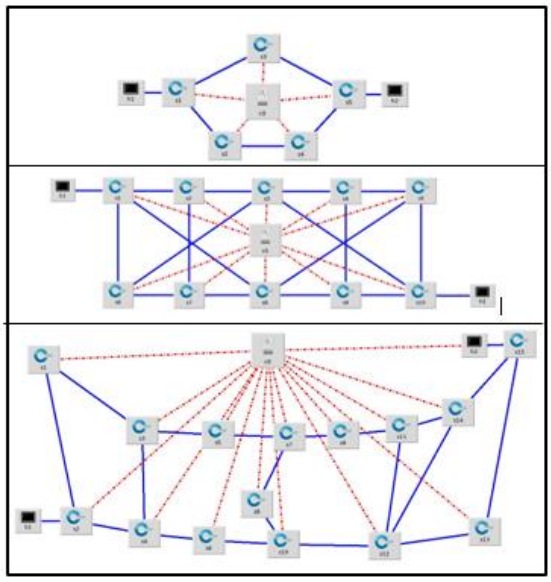
A. Perancangan Sistem

Perancangan Sistem akan dilakukan dengan melakukan simulasi pada aplikasi emulator jaringan bernama *mininet* yang terdapat *Operating system* berbasis *linux.* Dan menggunakan *Ryu* sebagai *Controller SDN* kemudian juga diperlukan perancangan Topologi dan Perancangan Algoritma *Routing* yaitu Algoritma *Dijkstra* dan Algoritma *Floyd Warshall* yang nantinya akan dilakukan beberapa pengujian dan dibandingkan untuk melihat manakah algoritma yang memiliki performa lebih baik pada jaringan *OpenFlow.* Alur perancangan sistem dapat dilihat berdasarkan dengan *flowchart* pada Gambar 3.



Gambar 3 *Flowchart* Sistem

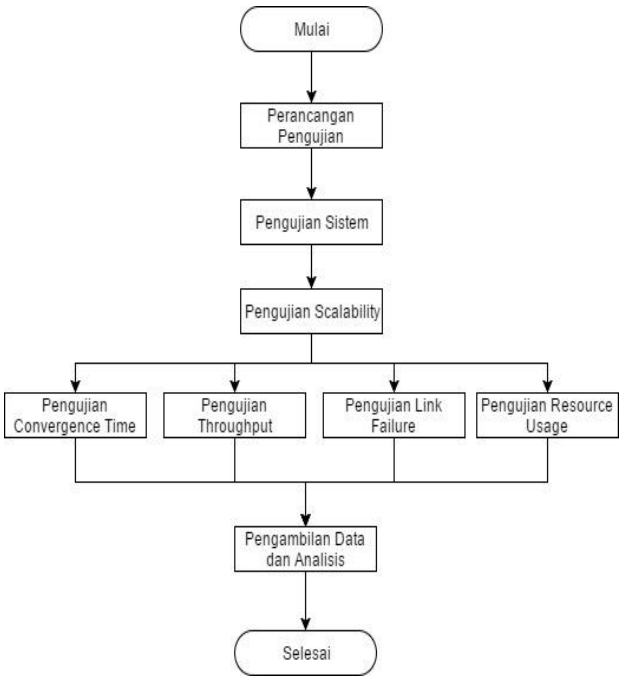
Sistem ini mensimulasikan jaringan berbasis *SDN* menggunakan *Ryu SDN framework* sebagai kontroler dengan menggunakan *NetworkX* sebagai perangkat lunak yang akan memanipulasi topologi jaringan pada jalur data menjadi sebuah graph untuk kemudian me-ngimplementasikan algoritma *routing* pada kontroler dengan mengakses *library* yang sudah tersedia pada *NetworkX* untuk penentuan jalur yang ada pada topologi yang sudah dibuat sedangkan untuk bagian jalur data menggunakan *emulator Mininet*, Terdapat 3 topologi yang diuji untuk menilai keberhasilan *routing* algoritma yang dilakukan, dan secara urut dinotasikan dengan TP1, TP2, dan TP3 dengan jumlah *switch* secara urut 5,10, dan 15 *switch*.yang terlihat pada Gambar 4. Dengan ketentuan *cost* bernilai 1 untuk setiap *link*



Gambar 4 *TP1, TP2, TP3*

3. Simulasi dan Analisis

Terdapat lima pengujian yaitu pengujian *scalability, convergence time, throughput, link failure,* dan *resource usage.* Pada gambar 5 terlihat alur pengujian dari sistem yang akan dibuat.



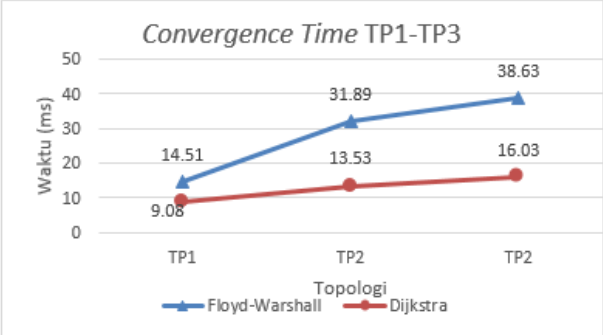
Gambar 5 *Flowchart* Pengujian Sistem

1) Pengujian *Scalability*

Pengujian *Scalability* akan dilakukan pada sistem sebelum dilakukan pengujian selanjutnya, pengujian ini akan dilakukan dengan menjalankan setiap Algoritma *routing* pada 3 macam topologi yang dengan jumlah switch berbeda-beda yaitu 5, 10, dan 15 *switch* pada setiap topologi. Kemudian dilakukan pengujian-pengujian selanjutnya.

2) Pengujian *Convergence Time*

Pengujian *convergence time* dilakukan untuk mengetahui waktu yang dibutuhkan controller untuk membentuk sebuah tabel *routing*. Pengujian dilakukan dengan cara mengirimkan paket *Internet Control Message Protocol* (ICMP) dengan menggunakan *command ping* dari *host* sumber menuju *host* tujuan pada topologi. Setelah mendapatkan *reply* atau balasan dari *host* tujuan, *controller* akan menampilkan jalur terpendek dan *convergence time* pada topologi tersebut.



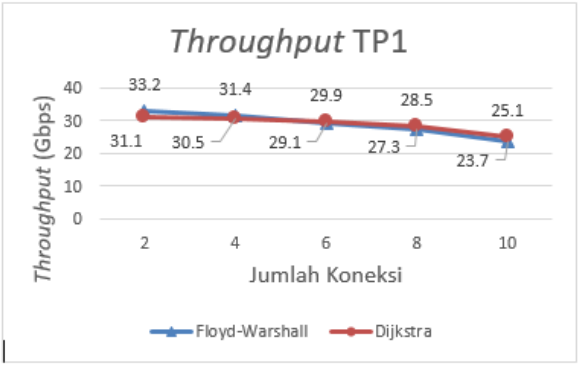
Gambar 6 Grafik pengujian *convergence time*

Pada gambar 6 merupakan rata-rata nilai dari hasil pengujian *convergence time* dilakukan sebanyak 10 kali pada setiap topologi. Dari hasil yang di dapat untuk setiap variansi jumlah *switch* pada setiap topologi akan mengalami perbedaan *convergence time* karena apabila dilihat dari hasil pengujian nya, *convergence time* bertambah seiring dengan penambahan *switch* pada topologi. Jika diurutkan berdasarkan jumlah *switch* pada topologi, maka TP1< TP2 < TP3, sama seperti perbandingan *convergence time* rata-rata dari topologi-topologi tersebut. Dari hasil pengujian juga dapat kita ketahui bahwa pada semua topologi yang sudah diujikan bahwa Algoritma *Dijkstra* lebih unggul dibandingkan dengan Algoritma *Floyd-Warshall* Karena memiliki *convergence time* lebih cepat dibandingkan dengan algoritma *Floyd-Warshall*.

3) Pengujian *Throughput*

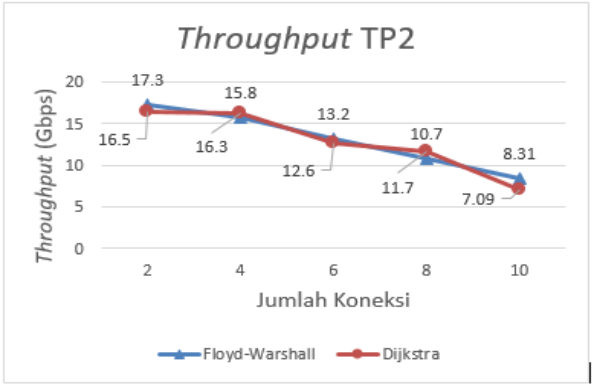
Pengujian *throughput* dilakukan untuk mengetahui kecepatan data yang diperoleh untuk mengirimkan paket ke *host* tujuan. Pengujian dilakukan dengan dengan menggunakan aplikasi *iperf* yang dijalankan pada dua buah *host* yang satu bertindak sebagi *server* dan yang lainnya

bertindak sebagai *client* yang akan mengirimkan paket data. Dengan dilakukan peningkatan jumlah koneksi *client* dari 2-10 *client* selama 30 detik. Untuk mengukur *throughput*, digunakan aplikasi *iperf* di antara h1 dan h2 dari masing-masing topologi. Di sini, h1 bertindak sebagai *server* dan h2 bertindak sebagai *client*. Pengujian *throughput* dilakukan dengan beban *traffic* TCP selama 30 detik untuk masing-masing topologi TP1-TP3 dengan dilakukan peningkatan jumlah koneksi *client* secara *parallel* dari 2 hingga 10 *client* secara bersamaan. Hasilnya berupa rata-rata selama 30 detik.



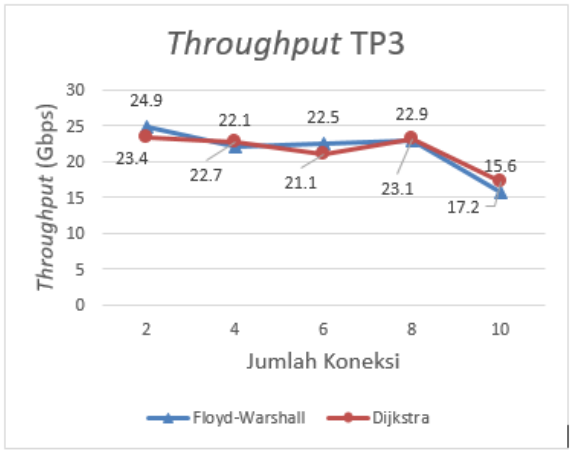
Gambar 7 Grafik pengujian *throughput* di TP1

Berdasarkan Gambar 7 hasil pengujian *throughput* pada topologi TP1, dapat terlihat bahwa terdapat penurunan *throughput* pada Algoritma *Dijkstra* ataupun *Floyd-Warshall* pada peningkatan jumlah koneksi *client.* Dengan *throughput* terbesar yaitu 30-35 Gbps terjadi pada saat jumlah koneksi sebanyak 2 *client.*



Gambar 8 Grafik pengujian *throughput* di TP2

Berdasarkan Gambar 8 hasil pengujian *throughput* pada topologi TP2, terjadi penurunan *throughput* seiring bertambahnya koneksi *client* dan apabila dibandingkan pada *throughput pada* topologi TP1, *throughput* pada topologi TP2 mengalami penurunan dengan *throughput* terbesarnya antara 15-20 Gbps pada saat jumlah koneksi *client* sebanyak *2*.

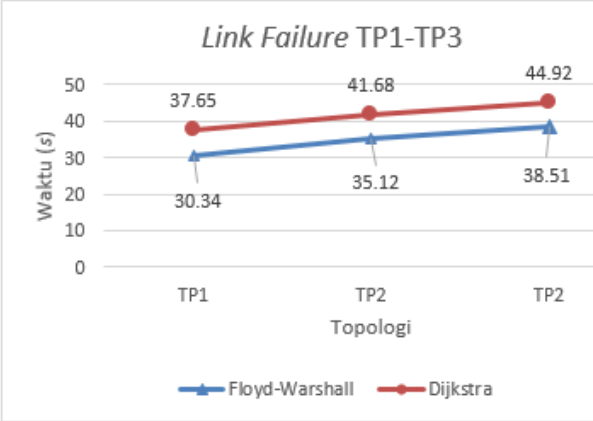


Gambar 9 Grafik pengujian *throughput* di TP3

Berdasarkan Gambar 10 hasil pengujian *throughput* pada topologi TP3 terjadi terjadi peningkatan *throughput* bila dibandingkan dengan *throughput* pada topologi TP2. Perbandingan *throughput* Algoritma *Dijkstra* dengan *Floyd-Warshall*, bila dilihat dan dibandingkan dari hasil pengujian *throughput* pada kedua algoritma, keduanya tidak memiliki perbedaan *throughput* yang cukup signifikan pada setiap topologi. Hal ini dikarenakan kedua Algoritma melalui jalur yang sama pada setiap topologi Karena keduanya termasuk dalam algoritma pencarian jalur terpendek atau *shortest path*.

4) Pengujian *Link Failure*

Pengujian *link failure* dilakukan untuk mengetahui seberapa cepat waktu yang diperlukan algoritma *routing* dalam menemukan jalur tependek yang baru jika terjadi pemutusan suatu *link* yang ada pada topologi. Seberapa cepat waktu yang diperlukan *controller* dalam memberikan informasi yang baru kepada *switch* untuk menemukan jalur terpendek yang baru, waktu yang diperlukan itu disebut *recovery time.* Pengujian dilakukan dengan melakukan pemutusan suatu *link* secara acak. Setelah dilakukan pemutusan *link* dilakukan pengiriman paket *ICMP* dengan menggunakan *command ping* dari suatu *host* ke *host* lain untuk melihat *recovery time* bagi *controller* untuk membuat jalur baru dengan *link* yang sudah terputus tersebut pada setiap topologi yang akan dilakukan pada penelitian ini.

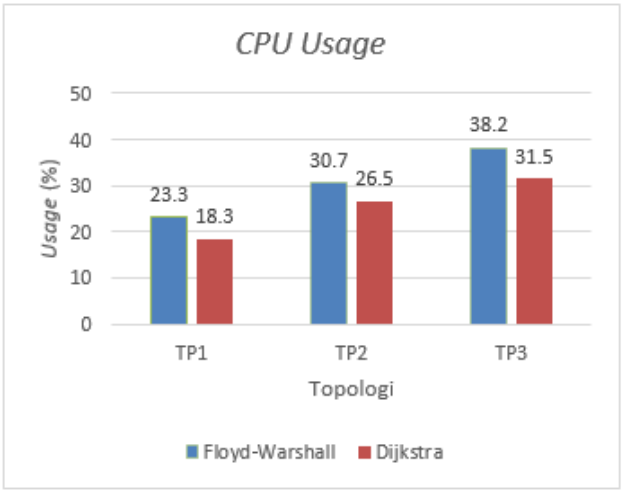


Gambar 10 Grafik pengujian *link failure*

Pada gambar 10 merupakan rata-rata nilai dari hasil dari pengujian *link failure* sama seperti hasil dari pengujian *convergence time* yang pengujiannya juga dilakukan sebanyak 10 kali pada setiap topologi. Sistem telah berhasil menemukan jalur terpedek yang baru setelah dilalukan pemutusan beberapa *link* pada setiap topologi dengan menggunakan Algoritma *Dijkstra* dan *Floyd-Warshall* dan sistem juga dapat menampilkan *recovery time*. Perhitungan *recovery time* berdasarkan waktu yang dibutuhkan oleh *controller* dalam menemukan jalur terpendek yang baru setelah dilakukan pemutusan *link*. Dari hasil pengujian dapat diketahui semakin banyak *switch* pada suatu topologi maka akan membutuhkan *recovery time* lebih lama untuk menemukan jalur terpendek yang baru pada suatu topologi. Dari hasil pengujian *link failure ini* juga dapat kita ketahui bahwa hasilnya berkebalikan dengan Hasil pengujian *convergence time*, pada pengujian *link failure* ini dapat dikatakan bahwa Algoritma *Floyd-Warshall* lebih unggul dibandingkan dengan Algoritma *Dijkstra* Karena memiliki *recovery time* yang lebih cepat.

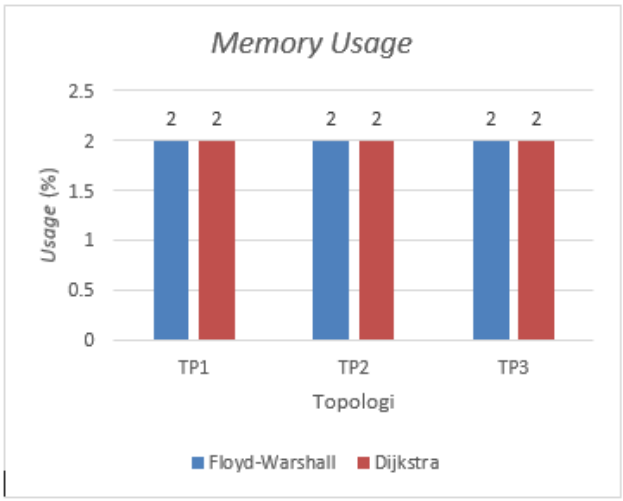
5) Pengujian *Resource Usage*

Pengujian *resource usage* dilakukan untuk mengetahui besar *CPU* dan *memory* yang digunakan oleh *controller* dalam sistem simulasi yang dibuat. Satuan yang digunakan adalah dalam persentase (%). Pengujian dilakukan dengan cara menjalankan perintah "*PS Aux*" pada *terminal* untuk melihat seberapa besar pemakaian *CPU* dan *memory* yang digunakan oleh suatu *program* atau aplikasi.



Gambar 11 Grafik pengujian *resource usage (CPU)*

Pada Gambar 11 merupakan hasil dari pengujian *resource usage* dapat dilihat nilai konsumsi *CPU* yang digunakan oleh *controller* untuk menjalankan *program routing*, menunjukan hasil yang siginifikan bertambah seiring dengan bertambahnya juga jumlah *switch* pada topologi. Hal ini dikarekanan, semakin banyak jumlah *switch* yang digunakan maka perhitungan *routing* pada setiap algoritma juga akan semakin banyak.



Gambar 12 Grafik pengujian *resource usage (Memory)*

Dari hasil pengujian yang telah dilakukan dapat diketahui bahwa Algoritma *Floyd-Warshall* membutuhkan Konsumsi *CPU* yang lebih besar dibandingkan dengan Algoritma *Dijkstra* pada setiap topologi. Di lain sisi jika dilihat pada jumlah konsumsi *memory* keduanya sama-sama membutuhkan *memory* sebesar 2% .

4. KESIMPULAN

Berdasarkan hasil pengujian dan analisis dari implementasi sistem ini, dapat ditarik kesimpulan dari penelitian ini, yaitu sebagai berikut.

1. Sistem telah mampu melakukan pencarian jalur terpendek untuk kedua algoritma pada setiap topologi.
2. Berdasarkan hasil pengujian *convergence time*, *convergence time* meningkat seiring dengan bertambahnya jumlah *switch* yang ada pada tiap topologi. Dengan hasil Algoritma *Dijkstra* lebih unggul Karena memiliki waktu lebih cepat dibandingkan Algoritma *Floyd-Warshall*.
3. Berdasarkan hasil pengujian *throughput*, pada kedua algoritma tidak terlalu memiliki perbedaan yang signifikan.
4. Untuk hasil pengujian dari *link failure* banyaknya *switch* pada setiap topologi berbanding lurus dengan *recovery time* yang diperlukan oleh *controller* untuk menentukan jalur yang baru, pada pengujian ini Algoritma *Floyd-Warshall* lebih unggul karena memiliki *recovery time* lebih cepat dibandingkan dengan Algoritma *Dijkstra*.
5. Untuk hasil dari pengujian *resource usage*, pada pengukuran *memory usage* kedua algoritma mengkonsumsi *memory* sama besar, namun pada pengukuran *CPU Usage*, Algoritma *Dijkstra* mengkonsumsi daya *CPU* lebih sedikit bandingkan dengan Algoritma *Floyd-Warshall*.

5. DAFTAR PUSTAKA

Astuto, B. N., Mendon¸ca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. 2014. *A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks.* IEEE Commu nications Society, Institute of Electrical and Electronics Engineers.

Dely, A. Kassler and N. Bayer. 2011. *OpenFlow for Wireless Mesh Networks*. Maui.

Jiang, H.-W. Huang, J.-H. Liao and S.-Y. Chen. 2014. *Extending Dijkstra's Shortest Path Algorithm for Software-Defines Networking.* Hsinchu.

Lantz, B. Heller. 2010. *A Network in a Laptop: Rapid Prototyping for Software-Defined Networks*. New Delhi.

McKeown, T. Anderson, H. Balakrishnan, G. Parulkar. 2008. *OpenFlow: Enabling Innovation in Campus Networks*.

Noviandi, Raden Aprian. 2007. Perbandingan Algoritma *Dijkstra* dan Algoritma *Floyd-Warshall* dalam Penentuan Lintasan Terpendek. Indonesia.

Saputra, Ihsan Aris., R.Sumani M., & Sofia Naning Hetiana. 2016. Uji Performasi Algoritma *Floyd-Warshall* pada Jaringan *Software Defined Network* (*SDN*). Indonesia.