IMPLEMENTASI ALGORITMA *SHORTEST PATH JOHNSON* UNTUK MEKANISME *ROUTE DISCOVERY* PADA SOFTWARE DEFINED NETWORK

Akbar Pandu Segara[1)], Royyana Muslim Ijtihadie[2)], dan Tohari Ahmad[3)]
[1, 2, 3)]Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia 60111
e-mail: akbarpandu94@gmail.com[1)], roy@if.its.ac.id[2)], tohari@if.its.ac.id[3)]

ABSTRAK

Software Defined Network adalah suatu arsitektur jaringan dengan paradigma baru yang terdiri atas control plane yang ditempatkan secara terpisah dengan data plane. Segala bentuk kontrol perilaku jaringan komputer dilakukan oleh control plane. Sementara itu data plane yang terdiri dari router atau switch menjadi perangkat untuk forwarding paket. Dengan model control plane secara terpusat, SDN sangat rentan terhadap kongesti karena model komunikasi one-to-many. Terdapat beberapa mekanisme untuk kontrol kongesti pada SDN, salah satunya memodifikasi paket dengan mengurangi ukuran paket yang dikirim. Tetapi ini dianggap kurang efektif karena waktu yang dibutuhkan akan lebih lama karena jumlah paket yang dikirim lebih sedikit. Hal ini mengharuskan administrator jaringan harus dapat mengkonfigurasi sebuah jaringan dengan protokol dan algoritma routing tertentu. Algoritma Johnson digunakan dalam menentukan rute untuk forwarding paket, Dengan sifat all-pair shortest path yang dapat diterapkan pada SDN untuk menentukan melalui rute mana paket akan diteruskan dengan membandingkan seluruh node yang terdapat pada jaringan. Hasil pengujian latency dan throughput algoritma Johnson dengan algoritma pembanding menunjukkan hasil yang baik dan perbandingan hasil uji coba algoritma Johnson masih lebih unggul. Hasil response time algoritma Johnson saat pertama kali melakukan pencarian rute lebih cepat dibanding algoritma OSPF konvensional dikarenakan dengan karakteristik algoritma all pairs shortest path yang menentukan rute terpendek dengan membandingkan semua pasang node pada jaringan.

Kata kunci: *Johnson, pencarian rute, routing, rute terpendek, SDN.*

IMPLEMENTATION OF JOHNSON'S SHORTEST PATH ALGORITHM FOR ROUTE DISCOVERY MECHANISM ON SOFTWARE DEFINED NETWORK

Akbar Pandu Segara[1)], Royyana Muslim Ijtihadie[2)], and Tohari Ahmad[3)]
[1, 2, 3)] Department of Informatics, Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia 60111
e-mail: akbarpandu94@gmail.com[1)], roy@if.its.ac.id[2)], tohari@if.its.ac.id[3)]

ABSTRACT

Software Defined Network is a network architecture with a new paradigm which consists of a control plane that is placed separately from the data plane. All forms of computer network behavior are controlled by the control plane. Meanwhile the data plane consisting of a router or switch becomes a device for packet forwarding. With a centralized control plane model, SDN is very vulnerable to congestion because of the one-to-many communication model. There are several mechanisms for congestion control on SDNs, one of which is modifying packets by reducing the size of packets sent. But this is considered less effective because the time required will be longer because the number of packets sent is less. This requires that network administrators must be able to configure a network with certain routing protocols and algorithms. Johnson's algorithm is used in determining the route for packet forwarding, with the nature of the all-pair shortest path that can be applied to SDN to determine through which route the packet will be forwarded by comparing all nodes that are on the network. The results of the Johnson algorithm's latency and throughput with the comparison algorithm show good results and the comparison of the Johnson algorithm's trial results is still superior. The response time results of the Johnson algorithm when first performing a route search are faster than the conventional OSPF algorithm due to the characteristics of the all pair shortest path algorithm which determines the shortest route by comparing all pairs of nodes on the network.

Keywords*: Johnson, routing, route discovery, shortest path, SDN.*

I. PENDAHULUAN

Derformansi suatu jaringan dapat dipengaruhi oleh berbagai faktor salah satunya konfigurasi dan metode manajemen jaringan. Pada jaringan konvensional konfigurasi perangkat jaringan dengan menggabungkan beberapa vendor yang berbeda dalam satu jaringan membutuhkan usaha tambahan, dikarenakan arsitektur jaringan yang begitu kompleks. *Software Defined Network* (SDN) merupakan paradigma baru arsitektur jaringan

http://dx.doi.org/10.12962/j24068535.v16i1.a590

yang menawarkan konsep *control plane* yang terpisah dengan *data plane* [1]. Pada *control plane* diimplementasikan sebuah *controller* yang secara terpusat berfungsi untuk melakukan *control* terhadap perilaku jaringan komputer. Sedangkan perangkat jaringan seperti *switch* atau *router* pada sisi *data plane* menjadi perangkat untuk *forwarding* paket. Namun dengan model *control plane* terpusat SDN sangat rentan akan terjadinya kongesti dikarenakan model komunikasi *one-to-many* [2].

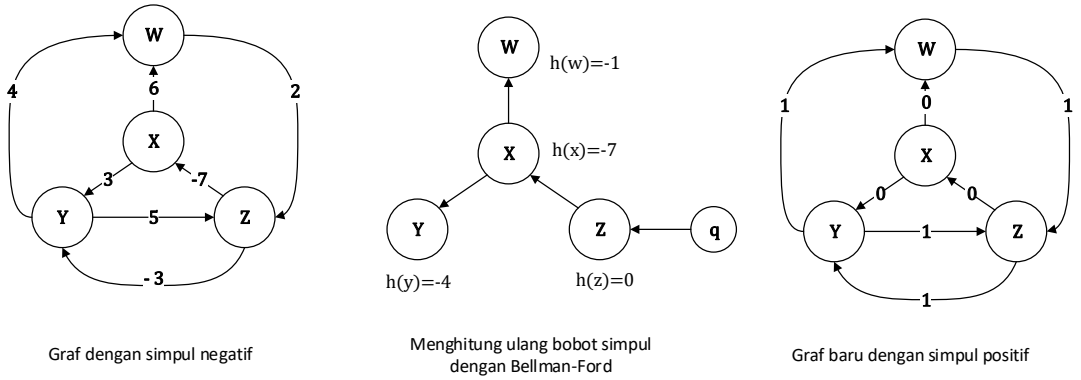
Menurut paper [3] dan [4] pada SDN terdapat beberapa mekanisme kontrol kongesti, salah satunya mekanisme deteksi kongesti pada jaringan dengan mengurangi ukuran pengiriman paket dari *source*. Akan tetapi, hal tersebut dirasa mengakibatkan waktu pengiriman akan menjadi lebih lama karena jumlah paket yang dikirim lebih sedikit. Hal ini mengakibatkan administrator jaringan harus mencari cara lain untuk melakukan manajemen jaringan. Tidak hanya pada arsitektur jaringan konvensional, pada arsitektur SDN protokol *routing* menjadi salah satu faktor yang mempengaruhi manajemen jaringan. Dengan mengkonfigurasi sebuah jaringan dengan protokol *routing* tertentu untuk menghindari rute yang sedang terjadi kepadatan.

Routing merupakan suatu hal yang sangat penting dalam membangun sebuah jaringan karena dalam proses *routing* terdapat pemilihan rute terbaik untuk mengirimkan paket dari pengirim ke penerima. Dalam *network routing* sering digunakan suatu algoritma untuk menentukan rute terpendek (*shortest path*). Hal ini berguna untuk mengurangi biaya pengiriman (*cost*) dan hilangnya paket (*packet loss*) saat proses pengiriman paket dari pengirim ke penerima. Algoritma *routing* diperlukan untuk membangun suatu protokol *routing* yang baik, dimana nantinya protokol tersebut dapat menentukan *shortest path* pada berbagai macam topologi jaringan tanpa harus melakukan konfigurasi ulang [5]. *Shortest Path Algorithm* (SPA) digunakan untuk menemukan jalur antara dua buah node pada sebuah *graph* sehingga diperoleh bobot *edge* yang minimum.

Terdapat beberapa jenis SPA yang umum digunakan dalam *Software Defined Network* yaitu *Algoritma Dijkstra, Algoritma Bellman-Ford, Algoritma Floyd Warshall*, dan *Algoritma Johnson*. Suatu Layanan IP menggunakan *Open Shortest Path First* (OSPF) dengan perhitungan yang berdasarkan pada algoritma *Dijkstra* yang menghitung *shortest path* dalam jaringan [6]. Dalam perkembangannya, SPA banyak diimplementasikan pada skenario pencarian rute (*route discovery*) terpendek pada *Software Defined Network.* Salah satunya yaitu algoritma *Johnson*, yang merupakan penggabungan dua buah algoritma *Single-Source Shortest Paths*, yaitu algoritma *Dijkstra* menentukan rute terpendek berdasarkan jarak antar node sedangkan Algoritma *Bellman-Ford* perhitungannya berdasarkan jumlah hop [7]. Karena merupakan penggabungan dari 2 algoritma *shortest path*, Algoritma *Johnson* memiliki keunggulan yaitu dengan sifat *all-pair shortest path* akan mencari *shortest path* berdasarkan semua pasangan *node* sehingga *runtime* akan lebih cepat. Perhitungan dari algoritma *Johnson* dimulai dengan menghitung *single-path* dari suatu node ke node lainnya menggunakan Algoritma *Bellman-Ford*. Setelah jarak itu dihitung maka kemudian memperbaharui bobot pada semua *edge* dengan menggunakan jarak yang dihitung antar sumber sebelumnya dengan menggunakan Algoritma *Dijkstra* [8].

Penelitian ini bertujuan untuk mengembangkan mekanisme *route discovery* untuk memilih rute alternatif menggunakan algoritma *routing johnson* untuk memilih rute alternatif pada *Software Defined Network.* Hal ini dirasa perlu karena pada SDN merupakan arsitektur jaringan dengan komunikasi *one-to-many* yang sangat rentan terhadap kongesti. Kongesti dapat menurunkan performa jaringan jika tidak ditangani dengan segera maka akan meningkatkan *delay* hingga turunnya *throughput* yang akan berdampak pada proses pengiriman paket. Oleh karena itu, algoritma *routing Johnson* dipilih sebagai usulan karena dirasa dapat melakukan pencarian rute dengan waktu eksekusi yang cepat. Modifikasi pada metode *route discovery* dengan menggunakan kemampuan dari algoritma *Johnson* untuk menentukan rute lain dengan *cost minimum* jika *latency* yang terjadi pada rute utama mengalami ketidakstabilan karena rute tersebut dilalui oleh pengiriman paket dari host lain. *Data plane* akan memberikan informasi pada *controller* terkait *traffic* yang terjadi pada *Openflow switch*. Kemudian *controller* akan merespon informasi dengan pemilihan rute alternatif disimpan pada *flow* tabel. Proses diteruskan dengan pengecekan paket, jika paket melewati rute yang sedang terjadi *congestion* maka *controller* akan mendeteksi adanya *delay* yang berisi informasi bahwa pada rute terpendek yang dilalui sedang padat dan paket akan dilewatkan melalui rute alternatif terpendek yang lalu lintas datanya tidak padat. Modifikasi algoritma *Johnson* akan diimplementasikan pada sisi *control plane* dengan basis *remote controller Ryu* menggunakan bahasa pemrograman *python*. Dan memanfaatkan kemampuan *OpenFlow switch* dalam menyaring informasi paket yang masuk pada sisi *data plane* sehingga dapat dengan cepat mendapatkan respon jika terjadi perubahan rute. Dengan demikian tidak ada pengurangan ukuran paket dan diharapkan dapat mengurangi *latency* dan mempertahankannya seminimal mungkin.

Makalah ini membahas tentang mekanisme *route discovery* pada arsitektur *Software Defined Network* dengan memanfaatkan algoritma *routing Johnson*. Sistematika penulisan makalah ini, bagian 1 membahas mengenai pendahuluan makalah, bagian 2 membahas mengenai kajian pustaka tentang topik yang dibahas pada makalah ini. Bagian 3 membahas tentang metodologi penelitian yang digunakan pada makalah ini. Bagian 4 membahas tentang hasil dan pembahasan. Bagian 5 membahas tentang kesimpulan yang ditarik dari makalah ini.



Gambar 1. Perhitungan graf algoritma Johnson [9].

II. KAJIAN PUSTAKA

A. *Software Defined Network*

Paradigma *Software Defined Network* dibuat untuk mengatasi keterbatasan suatu infrastruktur jaringan. Dengan konsep memisahkan *network control logic* dari *router* dan *switch* yang meneruskan *traffic.* Dengan memisahkan *control plane* dan *data plane*, *switch* menjadi perangkat untuk meneruskan paket, dan dilakukan *control* secara terpusat pada *network control logic*. Hal ini akan menyederhanakan konfigurasi pada sebuah jaringan dengan banyak perangkat jaringan dari vendor yang berbeda.

Arsitektur SDN tersusun dari tiga layer yang terdiri dari *application layer*, *control layer* dan *infrastructure layer*. *Application layer* berada pada lapisan paling atas, layer ini merupakan letak aplikasi SDN yang diprogram untuk berkomunikasi dengan *controller* SDN. Pada *control layer* terdapat sebuah SDN *controller* yang berfungsi menerima instruksi dari *layer application* dan secara *centralized* mengontrol sebuah infrastruktur jaringan baik yang sifatnya *centralized* maupun *decentralized* secara fisik. Sedangkan *infrastructure layer* adalah sekumpulan perangkat jaringan yang berfungsi untuk *forwarding* data. Dengan dipisahnya *control plane* dan *data plane* maka *switch* dan *controller* SDN dapat diimplementasikan antarmuka pemrograman. *Data plane* dapat dikontrol langsung oleh *controller* melalui *application programming interface* (API) [10].

B. *OpenFlow*

Pada arsitektur SDN, *open interface* dapat memprogram sebuah *software* berbasis *controller* (*control plane*) dan *network device* menjadi perangkat yang berfungsi untuk meneruskan paket (*data plane*) secara terpusat (*centralized*). Protokol SDN yang umum dipakai adalah protokol *OpenFlow*. *OpenFlow* mendukung pengguna untuk mengembangkan fungsi baru terhadap struktur network saat ini melalui sebuah *software*. *OpenFlow* mempunyai performansi keamanan yang bagus, manajemen yang mudah, *OpenFlow* juga berlaku untuk memanajemen jaringan virtual, menyederhanakan struktur jaringan, mengurangi biaya operasi dan konstruksi jaringan [11]. *OpenFlow* adalah salah satu implementasi dari *open interface*. *OpenFlow* memungkinkan akses langsung untuk memanipulasi *forwarding plane* pada perangkat jaringan seperti *switch* dan *router*. Protokol *OpenFlow* melakukan mekanisme komunikasi terhadap *switch* menggunakan *OpenFlow*. Secara *centralized*, *OpenFlow* mengontrol kerumitan jaringan ke dalam sebuah *controller software*, sehingga administrator dapat melakukan pengaturan jaringan melalui *controller* dengan mudah [12].

C. *Shortest Path Algorithm (SPA)*

SPA (*Shortest Path Algorithm*) digunakan untuk menemukan jalur antara dua buah node pada sebuah *graph* sehingga diperoleh bobot *edge* yang minimum. Untuk kelebihan dari SPA sendiri diantaranya untuk meminimalkan jarak menuju node tujuan dan untuk mengurangi *loss* pada distribusi *power flow*. Terdapat dua buah jenis SPA diantaranya *minimum spanning tree* dan *graph theory*. Salah satu SPA jenis *graph theory* yang bersifat *all pair shortest path* merupakan Algoritma *Johnson*, yaitu algoritma yang digunakan dalam menentukan *shortest path* berdasarkan semua pasangan *node*, penentuan *shortest path* cepat dan performansinya stabil [13].

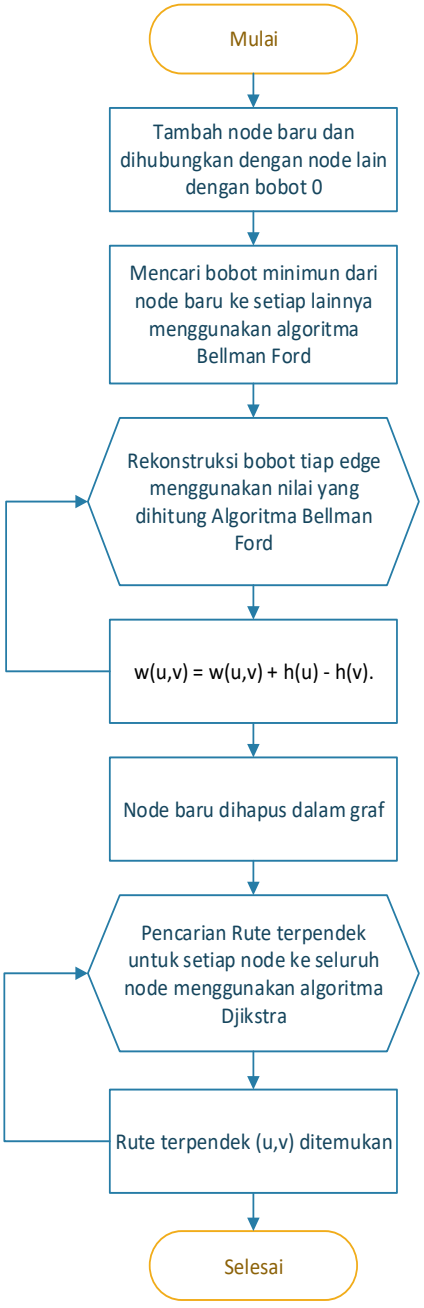
Dalam Algoritma *Johnson* terdapat beberapa langkah untuk menghitung jalur terpendek yang dapat dilihat pada Gambar 1 yaitu dengan menambahkan sebuah node baru *q* dengan nilai 0 pada sebuah *graph* dengan bobot negatif. *Node* baru *q* dengan jarak *source node* adalah 0 dikarenakan node *q* diasumsikan sebagai *source node* baru. Kemudian algoritma *Bellman-Ford* akan menghitung bobot minimum dari graph dengan node *q* sebagai titik awal. Setelah bobot minimum dihitung maka akan mendapatkan bobot *graph* yang telah diperbaharui dan mendapatkan bobot positif. Kemudian node *q* dihilangkan lalu algoritma *Dijkstra* akan menghitung rute terpendek dari masing-masing *node* pada *graph* yang telah diperbaharui. Algoritma *Bellman-Ford* digunakan untuk mengatur ulang input *graph* untuk menghilangkan bobot negatif pada tiap *edge* dan mendeteksi siklus negatif dari *graph*. Kemudian dari *graph* baru yang telah dibuat oleh algoritma *Bellman-Ford*. Kemudian Algoritma *Dijkstra* akan menghitung rute

dengan cost minimum dengan membandingkan seluruh pasangan simpul *node*. Algoritma *dijkstra* mencari rute dengan *cost* paling minimum antara titik satu dengan yang lain. Selain itu, algoritma ini juga menghitung total *cost* yang dari jalur lintasan terpendek [14].

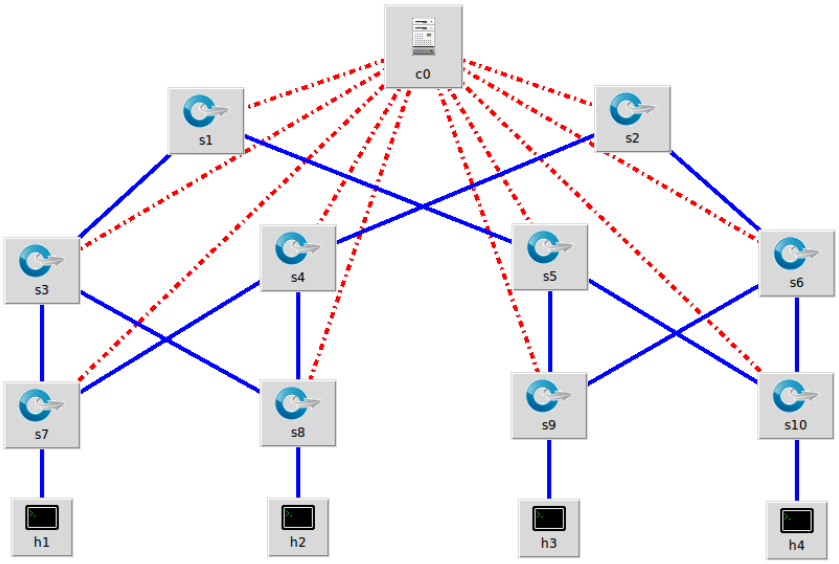
III. METODOLOGI PENELITIAN

Metode yang diusulkan adalah pemilihan rute paket pada saat pengiriman data, algoritma *routing* yang digunakan algoritma *Johnson* untuk menentukan *shortest path* saat pengiriman data. Untuk menentukan rute terpendek, pada Algoritma *Johnson* menambahkan *node* baru pada graf. Selanjutnya adalah merekonstruksi bobot pada semua *edge node* menggunakan Algoritma *Bellman-Ford* sebagai subrutin. Kemudian Algoritma *Dijkstra* digunakan untuk mencari rute terpendek dari tiap *node* ke semua *node* lain.

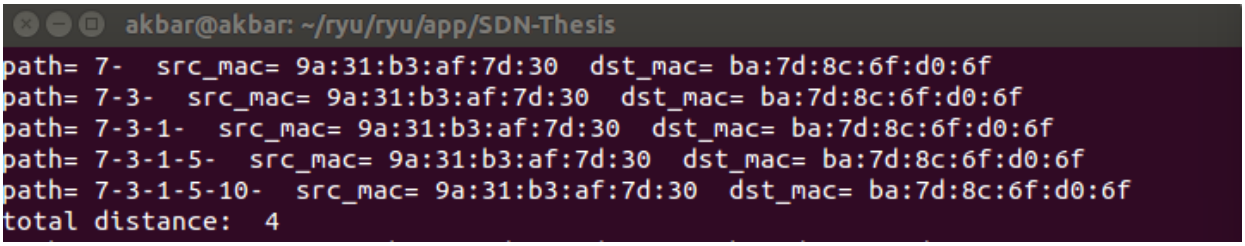
Adapun tahapan perancangan Algoritma *Johnson* dapat dilihat pada Gambar 2. Berdasarkan Gambar 2 yaitu mekanisme penentuan rute terpendek menggunakan Algoritma *Johnson* diawali dengan menambahkan sebuah *node* baru yang dikoneksikan pada setiap *node* lainnya dengan bobot 0. Kemudian menghitung bobot minimum (dengan *node* baru sebagai titik awal) ke setiap node lainnya dengan menjalankan Algoritma *Bellman-Ford*. Tahap selanjutnya adalah merekonstruksi bobot pada setiap *node* terdekat dengan persamaan 1, dimana *w* merupakan bobot dari suatu node, *u* adalah node awal atau sumber dan *v* merupakan node tujuan. Sedangkan *h* merupakan *hop*.



Gambar 2. Perancangan mekanisme *routing*.



Gambar 3. Topologi uji coba.



Gambar 4. *Log Ryu Controller* rute alternatif yang dipilih.

$$w'(u,v) = w(u,v) + h(u) - h(v). \quad (1)$$

$$D'(u,v) = D(u,v) + h(v) - h(u) \quad (2)$$

Lalu *node* baru tersebut dihapus dari topologi dan selanjutnya menghitung shortest path dari setiap *node* ke *node* lainnya dengan bobot yang diberikan menggunakan persamaan 2 dengan menggunakan Algoritma *Dijkstra*.

Topologi uji coba *fat-tree* yang dapat dilihat pada Gambar 3. Topologi ini dipilih karena sering digunakan pada implementasi *SDN* dalam skenario *data center* [15]. Selain itu topologi ini juga memiliki rute alternatif sehingga jika salah satu rute utama tidak tersedia maka *controller* masih dapat memilih rute lain yang tersedia. Dengan karakteristik algoritma *shortest path* maka algoritma akan menghitung jalur mana yang memiliki minimum *cost*. Algoritma ini akan diupload pada *remote controller* untuk kemudian *controller* akan mengatur konfigurasi dan informasi *switch* secara terpusat.

Mekanisme *routing* yang diusulkan ketika paket melewati sebuah *switch* akan selalu dilakukan pendeteksian *congestion* jika terjadi *congestion* pada jalur pengiriman, maka *data plane* akan menginformasikan *controller* untuk menggunakan algoritma *Johnson* sebagai penentuan rute yang akan dilalui paket untuk sampai ke tujuan dengan pertimbangan jalur alternatif dengan *cost* dan rute yang paling minimum. Kemudian *controller* akan menginformasikan kepada *data plane* agar informasi pemilihan jalur alternatif disimpan pada *flow* tabel. Kemudian proses diteruskan dengan pengecekan paket, apakah menyimpan informasi *congestion*. Jika paket tidak melewati jalur yang sedang terjadi *congestion* maka paket tersebut termasuk paket normal yang akan dikirimkan dari *source* ke *destination*. Namun jika paket melewati jalur yang sedang terjadi kepadatan maka *controller* akan mendeteksi adanya *delay* yang berisi informasi bahwa pada jalur tersebut sedang terjadi pengiriman paket oleh host lain dan paket akan dilewatkan melalui rute alternatif terpendek yang lalu lintas datanya tidak padat.

Selain algoritma *Johnson* penulis juga mencoba melakukan uji coba dengan algoritma *routing* B dengan menggunakan protokol OSPF konvensional yang berbasis algoritma *Dijkstra*. Algoritma ini digunakan sebagai pembanding untuk mengetahui perbandingan performansi algoritma *routing* untuk *route discovery*. Untuk hasil dan pembahasan uji coba kedua algoritma akan dibahas pada sub-bab berikutnya.

TABEL I
HASIL UJI COBA ROUTE DISCOVERY.

Node Asal	Node Tujuan	Rute Tersedia	Rute Terpendek	Rute Alternatif
H1	H4	7-4-2-6-10 7-3-1-5-10 7-3-1-5-9-6-10 7-3-8-4-2-6-10 7-4-2-6-9-5-10 7-4-8-3-1-5-10 7-3-8-4-2-6-9-5-10 7-4-8-3-1-5-9-6-10	7-4-2-6-10	7-3-1-5-10

IV. HASIL DAN PEMBAHASAN

Dalam penelitian ini, kami melakukan beberapa skenario pengujian. Parameter yang kami gunakan untuk uji coba adalah *route discovery*, *throughput*, *latency*, dan *response time* dengan membandingkan performansi algoritma *Johnson* dengan protokol OSPF yang berbasis algoritma *Dijkstra.* Skenario pengujian dengan menggunakan topologi *fat-tree* dengan 10 *switch*, 4 *host* dan 1 *controller*. Skenario uji coba akan dijelaskan pada masing-masing sub-bab dengan hasil sebagai berikut.

A. Route Discovery

Uji coba perbandingan kedua algoritma dilakukan *host source* melakukan pengiriman data menuju H4 dengan menggunakan *tool ping* dari *Host source* H1. *Host* H2 dan H3 juga melakukan pengiriman paket menuju H4 untuk memadati rute tertentu. Sebelum menggunakan *route discovery* seluruh paket akan melewati *switch* 7-4-2-6-10 yang telah dipilih oleh *controller* sebagai rute utama. Pada pengujian ini juga dilakukan pengiriman data dari *host* H2 menuju H4 dan mendapatkan hasil pencarian rute 8-4-2-6-10, dan *host* H3 menuju H4 dengan rute 9-6-10.

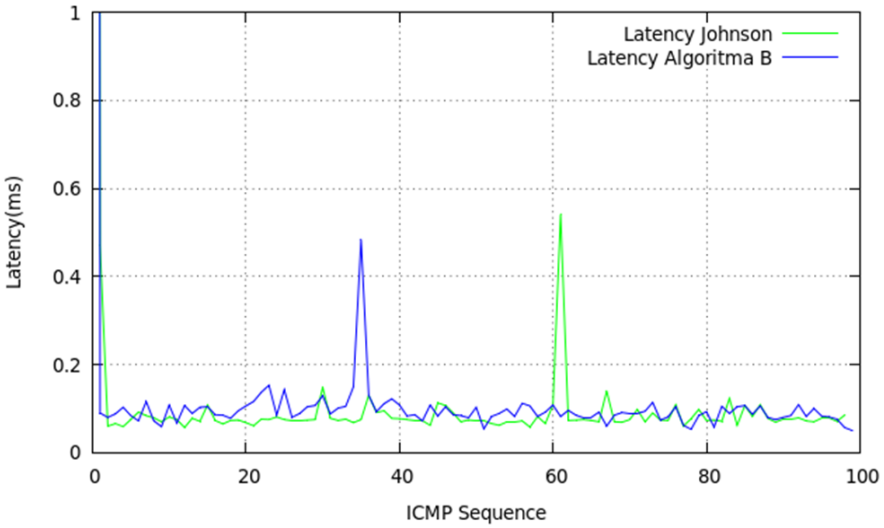
Pada skenario ini karena rute 4-2-6-10 telah dipadati oleh pengiriman paket dari host lain. melalui rute dan melalui *switch* tersebut. Karena pada dasarnya SPA akan memilih rute dengan *hop/cost* paling minim yang tersedia untuk menuju *host destination*. Kemudian saat menentukan rute alternatif maka *controller* akan menggunakan konsep *shortest path* dengan pertimbangan jalur dengan hop sama dengan jalur yang sebelumnya dilewati. *Controller* dapat menemukan rute alternatif lain yang jumlah hopnya sama dengan sebelumnya. Seperti yang dapat dilihat pada Gambar 4, dari hasil uji coba diatas *controller* dapat menemukan rute alternatif pada topologi *fat-tree* yang sama dengan rute sebelumnya sehingga paket dilewatkan rute alternatif yang tersedia yaitu melalui switch 7-3-1-5-10. Untuk tabel pencarian rute dapat dilihat pada Tabel I.

B. Latency

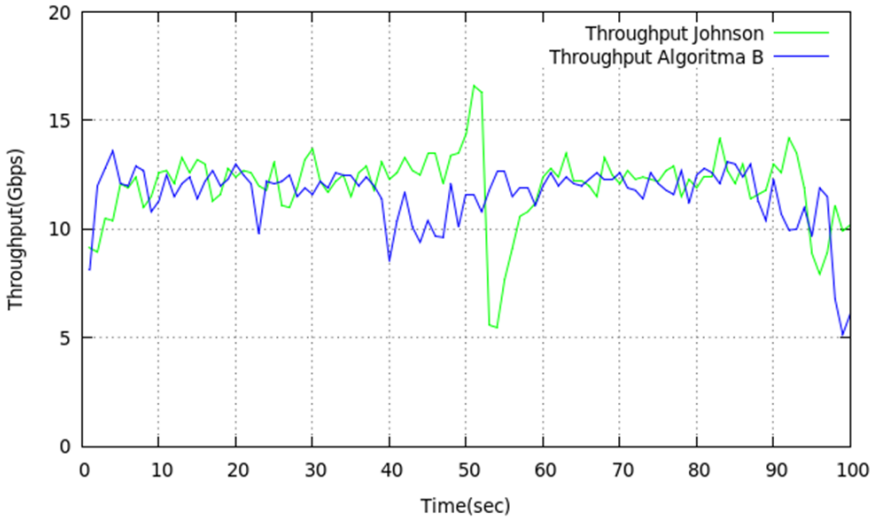
Pada pengujian *latency* dilakukan pengujian dengan melakukan perbandingan dua buah algoritma dalam melakukan *route discovery*. Pada skenario ini *host* H1, akan melakukan pengiriman data menuju *host destination* H4 dengan ukuran maksimal paket yang dikirim sebesar 20000 byte untuk memberikan beban pada saat pengiriman paket. H2 dan H3 juga akan melakukan pengiriman yang akan memadati rute sehingga akan terjadi *congestion* pada rute tertentu. Pengiriman dilakukan sebanyak 100 kali menggunakan tool *ping*. Hasil uji coba dapat dilihat pada Gambar 5. Pada Tabel II didapatkan perbandingan hasil pengujian dari *latency* dengan kedua algoritma dari minimal hingga maksimal *latency* dan standar deviasi yang dihasilkan pada saat pengiriman paket. Hasil uji coba rata-rata *latency* yang dihasilkan oleh algoritma *Johnson* sebesar 0.236 ms sedangkan *Dijkstra* 0.304 ms.

TABEL II
HASIL UJI COBA PERBANDINGAN LATENCY.

Algoritma *Johnson*				Algoritma B			
Min (ms)	Avg (ms)	Max (ms)	Mdev (ms)	Min (ms)	Avg (ms)	Max (ms)	Mdev (ms)
0.059	1.596	14.9	4.676	0.059	2.165	20.890	6.579
0.057	0.076	0.108	0.013	0.067	0.092	0.108	0.014
0.061	0.073	0.081	0.005	0.080	0.112	0.153	0.025
0.068	0.091	0.148	0.027	0.088	0.151	0.484	0.118
0.062	0.082	0.113	0.017	0.072	0.089	0.108	0.013
0.058	0.069	0.082	0.007	0.054	0.090	0.112	0.017
0.069	0.129	0.541	0.146	0.060	0.086	0.108	0.013
0.060	0.082	0.109	0.015	0.053	0.084	0.114	0.018
0.062	0.085	0.123	0.021	0.058	0.090	0.107	0.016
0.070	0.076	0.085	0.005	0.050	0.080	0.109	0.017
Avg 0.063	Avg 0.236	Avg 1.629	Avg 0.493	Avg 0.064	Avg 0.304	Avg 2.229	Avg 0.683



Gambar 5. Grafik perbandingan *latency*.

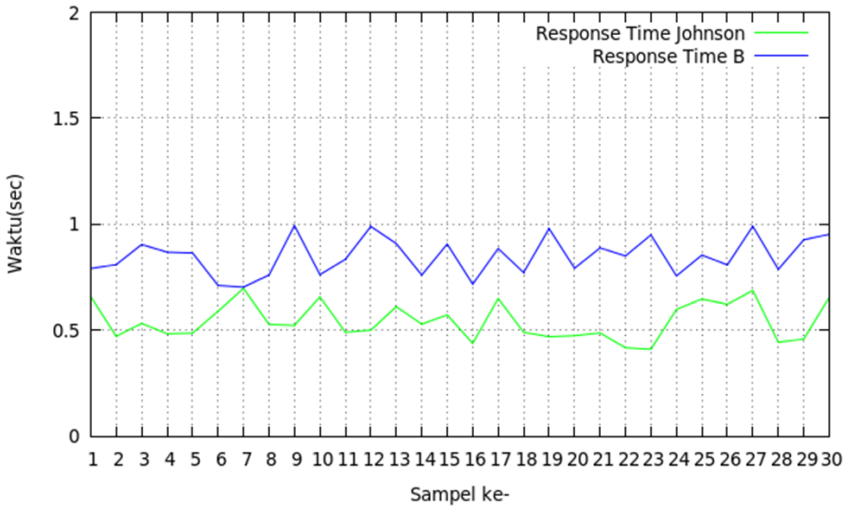


Gambar 6. Grafik perbandingan *throughput*.

Nilai maksimal *latency* sebesar 14.9 ms untuk *Johnson* dan 20.89 ms untuk *Dijkstra*, hal ini dikarenakan pengaruh dari *response time* pada algoritma *routing* untuk mencari rute baru pada saat pengiriman paket pertama, sehingga waktu yang dibutuhkan cenderung lebih lama karena *controller* memerlukan waktu untuk mengadaptasi rute yang baru. Sedangkan pada Protokol OSPF konvensional dengan dasar Algoritma *Dijkstra* sebagai *route discovery*. Rata-rata *latency* yang dihasilkan sebesar 0.304 ms. Rata-rata *latency* yang dihasilkan lebih tinggi dari algoritma *Johnson*. *Latency* pada pengiriman paket yang dihasilkan tergolong rendah dan stabil karena rute yang dilalui tidak dilewati oleh paket lain. Sehingga dapat disimpulkan mekanisme *route discovery* dengan dapat mengurangi *latency* dan mempertahankan *latency* agar lebih stabil saat pengiriman paket. Ada kemungkinan jika *controller* tidak dapat menemukan rute alternatif yang sama dengan rute sebelumnya dan terpaksa melalui rute yang lebih jauh maka *latency* yang dihasilkan akan semakin tinggi.

C. Throughput

Pada uji coba *throughput* dilakukan pengujian dengan melakukan perbandingan performansi dua buah algoritma *routing*. Pada skenario ini *host* H1 bertindak sebagai *client* dan *host* H4 bertindak *server*. Uji coba dilakukan dengan waktu 100 detik dengan interval pengiriman 1 detik sekali menggunakan tool *iperf*. Pada hasil uji coba didapatkan rata-rata *throughput* mencapai 12.0 Gbites/sec. Pada hasil uji coba dengan algoritma pembanding, algoritma *Dijkstra* menghasilkan rata-rata *throughput* 11.6 Gbps. *Throughput* rata-rata yang dihasilkan algoritma *Johnson* pemilihan rute alternatif masih lebih unggul. Algoritma OSPF konvensional menunjukkan performa yang cukup baik ditandai dengan minimnya throughput yang turun secara signifikan dalam uji coba, walaupun secara rata-rata *throughput* mengalami penurunan dibanding algoritma *Johnson*. Sehingga dapat disimpulkan *route discovery* dengan algoritma *shortest path* dapat meningkatkan kecepatan *throughput* jika rute alternatif yang dipilih tepat dan sesuai. Untuk grafik perbandingan masing-masing algoritma dapat dilihat pada Gambar 6.



Gambar 7. Grafik perbandingan *response time*.

TABEL III
HASIL UJI COBA RESPONSE TIME.

Sampel ke-	Algoritma *Johnson* (sec)	Algoritma B (sec)
1	0.659	0.792
2	0.472	0.810
3	0.533	0.905
4	0.484	0.869
5	0.487	0.865
6	0.590	0.712
7	0.698	0.704
8	0.528	0.761
9	0.524	0.994
10	0.658	0.763
11	0.491	0.835
12	0.501	0.990
13	0.612	0.909
14	0.529	0.761
15	0.573	0.907
16	0.439	0.718
17	0.650	0.886
18	0.491	0.773
19	0.470	0.980
20	0.475	0.793
21	0.488	0.889
22	0.418	0.851
23	0.411	0.950
24	0.599	0.757
25	0.648	0.855
26	0.623	0.809
27	0.688	0.991
28	0.444	0.788
29	0.459	0.927
30	0.655	0.953
Rata-rata	0.543	0.850

D. Response Time

Response time merupakan waktu yang dibutuhkan algoritma *routing* saat pertama kali melakukan pengiriman paket. Dari hasil uji coba *response time* pada metode yang dilakukan sebanyak 30 kali yang hasilnya dibandingkan dengan protokol OSPF yang berbasis algoritma *Dijkstra*. Hasil uji coba dapat dilihat pada Tabel III. Pada Gambar 7 dapat dilihat performansi algoritma *Johnson* lebih cepat dibanding algoritma *Dijkstra* saat pertama kali melakukan pengiriman paket. Dengan rata-rata waktu yang dibutuhkan algoritma *Dijkstra* selama 0.850 detik dengan waktu paling lama yaitu pada 0.994 detik. Sedangkan algoritma *Johnson* membutuhkan waktu rata-rata 0.543 detik dengan waktu paling lama 0.698 detik. Perbedaan yang terjadi pada pengujian dengan jumlah *switch* yang lebih banyak menunjukkan performansi algoritma *Johnson* lebih cepat dengan penurunan 17.8% dibandingkan algoritma *Dijkstra*. Algoritma *Johnson* dengan sifat *all pair shortest path* yang membandingkan semua pasang *node* terlebih dahulu. Sedangkan algoritma *Dijkstra* merupakan algoritma *greedy* dimana algoritma ini akan mencari nilai sementara antar *node* pada setiap langkahnya. Hal ini dapat disimpulkan performa algoritma

Johnson lebih baik saat melakukan pertama kali mengirimkan paket dikarenakan memiliki *response time* lebih cepat.

V. KESIMPULAN

Kesimpulan yang didapat berdasarkan hasil uji coba, pembahasan dan diskusi dapat dikatakan Algoritma *Johnson* dapat diimplementasikan dan berjalan cukup baik pada simulator *mininet* dan *controller Ryu* untuk *route discovery* dengan menentukan *shortest path* pada *Software Defined Network*. Berdasarkan parameter yang diuji diperoleh hasil pada penentuan rute host dapat memilih rute dengan jumlah *hop/switch* dan *cost* paling kecil. Pada pengujian *latency* algoritma *Johnson* lebih unggul. Pada pengujian *throughput* algoritma *Johnson* masih lebih unggul. Sehingga dapat disimpulkan algoritma *Johnson* dapat meningkatkan performansi *throughput*. *Response time* yang didapatkan saat algoritma *Johnson* setelah menentukan rute yang akan dilalui juga lebih cepat dibandingkan algoritma pembanding. Algoritma *Johnson* dengan sifat *all pair shortest path* yang membandingkan semua pasang node terlebih dahulu jika akan mencari rute terpendek. Sedangkan algoritma *Dijkstra* merupakan algoritma *greedy* dimana algoritma ini akan mencari nilai sementara antar *node* pada setiap langkahnya. Hal ini dapat disimpulkan Algoritma *Johnson* memiliki *response time* cukup yang baik saat melakukan pengiriman paket pertama kali dan juga akan performansinya cukup baik dibandingkan dengan algoritma pembanding *Dijkstra*.

DAFTAR PUSTAKA

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, dan S. Uhlig, "Software-Defined Networking : A Comprehensive Survey," dalam *Proc. IEEE*, vol. 103, no. 1, hal. 14–76, 2015.

[2] W. Chen, F. Ren, J. Xie, C. Lin, K. Yin, dan F. Baker, "Comprehensive Understanding of TCP Incast Problem," dalam *Proc. IEEE Conf. Comput. Commun.*, hal. 1688–1696, 2015.

[3] F. H. Saputra dan R. M. Ijtihadie, "Survei mekanisme congestion control pada transmission control protocol di software defined network," *JUTI J. Ilm. Teknol. Inf.*, vol. 6, no. 1, hal. 1–9, 2018.

[4] E. Auparay dan R. M. Ijtihadie, "Replikasi data menggunakan detection controller module untuk mencegah congestion di data center," *JUTI J. Ilm. Teknol. Inf.*, vol. 16, hal. 10–17, 2018.

[5] H. Xin, "Introduction of Centralized and Distributed Routing Protocols," dalam *Proc. Int. Conf. Electron. Commun. Control*, hal. 2698–2701, 2011.

[6] M. Grabowski, B. Musznicki, dan P. Zwierzykowski, "Review and Performance Analysis of Shortest Path Problem Solving Algorithms," *Int. J. Adv. Softw.*, vol. 7, no. July, hal. 20–30, 2014.

[7] A. Nastiti, A. Rakhmatsyah, dan M. A. Nugroho, "Link Failure Emulation with Dijkstra and Bellman-Ford Algorithm in Software Defined Network Architecture (Case Study : Telkom University Topology)," dalam *Proc. 2018 6th Int. Conf. Inf. Commun. Technol.*, vol. 0, no. c, hal. 135–140, 2018.

[8] D. B. Johnson, "Efficient Algorithms for Shortest Paths in Sparse Networks," *J. Assoc. Comput. Mach.*, vol. 24, no. 1, hal. 1–13, 1977.

[9] Brilliant, "Johnson's Algorithm," *brilliant.org*, 2019. [Online]. Available: https://brilliant.org/wiki/johnsons-algorithm/. [Diakses: 22-Oct-2019].

[10] H. Kim dan N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Communications Magazine*, no. February, hal. 114–119, 2013.

[11] A. Prajapati, A. Sakadasariya, dan J. P. Computer, "Software Defined Network : Future of Networking," dalam *Proc. 2018 2nd Int. Conf. Inven. Syst. Control*, no. Icisc, hal. 1351–1354, 2018.

[12] S. Azodolmolky, *Software Defined Networking with OpenFlow*, First Edit. Birmingham, Mumbai: Packt Publishing Ltd., 2013.

[13] B. . R. S Anitha, "Network Reconfiguration for Loss Minimization by Using Johnson's Algorithm," dalam *Proc. 2018 4th Int. Conf. Electr. Energy Syst.*, hal. 680–684, 2018.

[14] H. M. Abu-Ryash dan A. Tamimi A., "Comparison Studies for Different Shortest Path Algorithms," *Int. J. Comput. Technol.*, vol. 14, no. May, hal. 5980–5986, 2015.

[15] A. M. Abdelmoniem dan B. Bensaou, "Enforcing Transport-Agnostic Congestion Control in SDN-based Data Centers," dalam *Proc. IEEE 42nd Conference on Local Computer Networks*, 2017, hal. 120–136.