Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 3, Maret 2019, hlm. 2702-2710 http://j-ptiik.ub.ac.id

Perbandingan Routing Ulang Pada Algoritme Dijkstra dan Floyd-Warshall Dalam Mengatasi Link Failure Pada Arsitektur SDN

Risailin Dwi Jaka Fauzi[1], Rakhmadhany Primananda[2], Widhi Yahya[3]

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

Abstrak

Terdapat 2 (dua) jenis *routing*, *static routing* dan *dynamic routing*. *Static routing* merupakan jenis *routing* yang dilakukan secara manual, sedangkan pada *dynamic routing* merupakan jenis *routing* yang lebih melakukan proses *routing* secara otomatis. Salah satu arsitektur jaringan yang menerapkan *dynamic routing* saat ini adalah SDN, dalam melakukan proses *routing* terdapat beberapa masalah salah satunya adalah *link failure*, dimana kegagalan pada suatu jalur ini dapat mempengaruhi pengiriman data dalam jaringan. Saat ini banyak jenis algoritme routing seperti Algoritme Dijkstra, cara algoritme Dijkstra dalam menemukan jalur terpendek dengan melakukan pengecekan pada setiap jalur sampai menghasilkan jalur terbaik. Sedangkan Floyd-Warshall merupakan algoritme yang lebih dinamis dalam melakukan pencarian jalur terpendek tidak terpaku pada nilai cost yang paling rendah saja, tetapi juga mempertimbangkan konsekuensi kedepannya pada setiap langkah yang akan diambil. Untuk mengetahui perbandingan performansi peneliti melakukan perbandingan pada proses routing ulang menggunakan algoritme *Djikstra* dan *Floyd-Warshall* untuk mengetahui mana yang lebih cepat dalam mengatasi *Link Failure* pada *SDN*. Dari hasil perbadingan *convergence time* algoritme Djikstra dan Floyd-Warshall tersebut algoritme Dijkstra memiliki hasil yang lebih baik yaitu untuk skenario 1 0ms, skenario 2 16.401ms, dan skenario 3 17.200ms dibandingkan dengan algoritme Floyd-Warshall yaitu untuk skenario 1 0ms, skenario 2 19.803ms, dan skenario 3 20.401ms.

Kata kunci: *routing, link failure,* SDN, *dijkstra* dan *floyd-warshall*

Abstract

There are 2 (two) types of routing, static routing and dynamic routing. Static routing is a type of routing that will done manually, while dynamic routing is a type of routing that is more automatic routing. One of network architecture that implements dynamic routing at this time is SDN, in the routing process there are several problems, one of that problem is link failure, where failure on a path can affect data sending process in the network. At this time there are many types of routing algorithms such as Dijkstra's Algorithm, Dijkstra's algorithm way to finding the shortest path by checking each path until it produces the best path. While Floyd-Warshall is more dynamic algorithm in searching for the shortest path not just fixed on the lowest cost value, but also considering the future consequences of each step that will be taken. To know about performance comparison the researcher made a comparison on the re-routing process using the Djikstra and Floyd-Warshall algorithms to find out which ones were faster in overcoming the link failure at SDN. From the comparison of the Djikstra and Floyd-Warshall algorithm's convergence time, Dijkstra's algorithm has better results i.e for scenario 1 0ms, scenario 2 16.401ms, and scenario 3 17.200ms compared to Floyd-Warshall algorithm, for scenario 1 0ms, scenario 2 19.803 ms, and scenario 3 20.401ms.

Keywords: *routing, link failure,* SDN, *dijkstra and floyd-warshall*

1. PENDAHULUAN

Routing merupakan mekanisme penjaluran sebuah data dalam suatu jaringan. Terdapat 2 (dua) jenis *routing*, yaitu : *static routing* dan *dynamic routing*, static routing merupakan sebuah routing yang memiliki tabel routing yang didapatkan setelah administrator jaringan melakukan pengaturan pada router yang dipakai, rute

yang didapatkan juga diatur secara manual yang menyebabkan jika rute yang dipakai mengalami perubahan maka administrator jaringan harus melakukan update rute secara manual. Sedangkan *dynamic* routing merupakan sebuah routing yang lebih bersifat otomatis dimana pengaturan pada dynamic routing hanya dilakukan sekali, setelah itu dynamic routing akan mengerjakan pencarian jalur serta melakukan update jalur secara otomatis juga (Yunus, 2010). Salah satu arsitektur jaringan yang mendukung dynamic routing saat ini salah satunya adalah *Software Defined Network*. Dalam jaringan SDN terdapat beberapa permasalahan yang dapat terjadi saat proses routing dijalankan salah satunya yaitu *link failure. Link failure* merupakan suatu masalah yang terjadi disaat melakukan proses *routing*, dimana terjadi suatu kegagalan *link*. Jika jalur yang menghubungkan antara *switch* pada jaringan terjadi *link down* maka akan mempengaruhi proses pertukaran data dan jalur tersebut tidak akan dapat dilewati. Saat terjadi *down* pada jalur yang menghubungkan itu maka seharusnya *routing* protokol dapat langsung melakukan *update* pada jalur lain yang dapat digunakan dengan tetap memilih jalur yang terpendek lainnya (Aprilianingsih, et al., 2017).

Didalam membangun sebuah jaringan, pemilihan serta perancangan *routing* protocol yang tepat akan membuat sistem jaringan dapat menentukan jalur terbaik dalam pengiriman setiap paket data dari pelanggan ke tujuan serta dapat lebih meningkatkan kualitas layanan dari jaringan tersebut. (Joestin, et al., 2015). Walaupun didalam *Software Defined Network* yang menerapkan *dynamic routing* sudah dapat mengatasi permasalahan disaat terdapat gangguan dalam jaringan, tetapi dibutuhkan routing protocol yang tepat untuk dapat mendukung proses pencarian jalur didalam arsitektur tersebut.

Seiring perkembangan teknologi saat ini ada banyak algoritme *routing protocol* beberapa contohnya adalah Algoritme *Dijkstra* dan juga Algoritme *Floyd-Warshall,* kedua algoritme tersebut merupakan contoh dari algoritme yang menerapkan *link*-state *routing* protocol, dimana dalam *dynamic routing* algoritme yang menerapkan *link-state* dapat menemukan rute lebih cepat dibandingkan dengan algoritme yang tidak menerapkan *link-state*. Algoritme Dijkstra merupakan algoritme rakus (*greedy*) bukan dalam hal yang negatif, tetapi greedy disini merupakan cara algoritme Dijkstra dalam menemukan jalur terpendek dengan melakukan pengecekan pada setiap jalur sampai menghasilkan jalur dengan bobot yang paling kecil (Pradnyana, 2010). Berbeda dengan Dijkstra, Floyd-Warshall merupakan algoritme yang lebih dinamis. Dinamis ini dimaksudkan bahwa algoritme Floyd-Warshall dalam melakukan pencarian jalur terpendek tidak hanya terpaku pada satu parameter yaitu nilai cost yang paling rendah dalam setiap langkah yang diambil, tetapi juga mempertimbangkan konsekuensi kedepannya pada setiap langkah yang akan diambil. Karena itu algoritme Floyd-Warshall dianggap lebih baik dalam memberikan solusi yang terbaik.

Berdasarkan penelitian sebelumnya yaitu ”Analisis *Fail Path* Pada Arsitektur *Software Defined Network* Menggunakan *Dijkstra Algorithm*” (Aprilianingsih, et al., 2017), dari penelitian tersebut menerapkan *Dijkstra Algorithm* sebagai protokol *routing* yang dipakai, pada penelitian ini menghasilkan bahwa algoritme dijkstra dapat mengatasi permasalahan *link failure* pada Software Defined Network dengan melakukan *routing* ulang. Lalu berdasarkan penelitian ” *Demonstration Of Single Link Failure Recovery Using Bellman-Ford and Dijkstra Algorithm in Sdn*” (Waleed, et al., 2017), Dalam penelitian tersebut dilakukan perbandingan antara algoritme Bellman-Ford dengan algoritme Dijkstra dalam melakukan proses *routing* ulang untuk menyelesaikan permasalahan *link* failure. Selanjutnya ”Analisis Perbandingan Performansi Algoritme Floyd-Warshall dan Algoritme Johnson untuk Penentuan Rute Terpendek pada Software Defined Network” (Firdaus, et al., 2018), dimana pada penelitian ini menghasilkan bahwa pada

kondisi dimana *switch* memiliki kerapatan yang tinggi Floyd-Warshall memiliki keunggulan yang lebih dibandingkan dengan algoritme Johnson.

Dari penelitian sebelumnya menyimpulkan bahwa *Dijkstra Algorithm* dan *Floyd-Warshall* memiliki skema dalam melakukan *routing* ulang yang lebih baik untuk mengatasi *link failure*. Berdasarkan permasalahan diatas, maka akan dibuat sebuah simulasi pengujian untuk mengatasi *link failure* yaitu dengan melakukan pemutusan pada salah satu jalur *switch* menggunakan *link down*, ketika *link failure* terjadi maka jalur tersebut tidak akan dapat digunakan. Untuk menyelesaikan permasalahan ini akan dilakukan pencarian jalur terpendek lain atau alternatifnya dengan melakukan *routing* ulang menggunakan algoritme *routing Dijkstra* dan *Floyd-Warshall.* Waktu untuk menemukan jalur alternatif atau *convergence* ini akan menjadi dasar perbadingan antara kedua algoritme tersebut untuk menemukan mana algoritme yang lebih baik. Dari penelitian yang akan dilakukan ini maka diharapkan permasalahan *link failure* dapat diatasi dengan skema *routing* ulang dengan menggunakan algoritme *Dijkstra* dan *Floyd-Warshall* dan dapat mengetahui hasil perbandingan diantara algoritme tersebut yang memiliki performa dalam melakukan *routing* ulang yang lebih baik berdasarkan parameter *convergence time*-nya.

2. DASAR TEORI

2.1 Djikstra Algorithm

Algoritma Dijkstra adalah salah satu algoritma routing protocol yang termasuk dalam *single source shortest path* yang digunakan untuk menentukan jalur terpendek melalui perhitungan jumlah bobot terkecil pada node awal ke node tujuan. Algoritma ini diberi nama sesuai dengan penemunya yaitu ilmuwan komputer Belanda bernama Edsger Dijkstra. Algortima ini menerapkan prinsip Greedy yang memecahkan masalah lintasan terpendek untuk sebuah graf berarah dengan bobot sisi yang tidak negative. Bobot tersebut adalah bilangan positif jadi tidak dapat dilalui oleh node negative. Namun jika demikian maka penyelesainnya akan diberikan nilai infiniti atau tak hingga. Pada algoritma ini node digunakan karena algoritma Dijkstra menggunakan graf berasah dalam penentuan rute lintasan terpendek. Berikut *pseudocode* dari algortima Dijkstra. (Pradnyana, 2010).

Pseudocode algoritme Dijkstra

```
1   { Algoritma Dijkstra }
2
3   procedure Dijkstra
4
5         (   input m: matriks,   a:
    integer {simpul awal} )
6
7   {mencari lintasan terpendek dari
    simpul awal a ke semua simpul
    lainya Masukan : matriks
    ketetanggaa (m) dari graf
    berbobot G dan simpul awal a
    Keluaran: lintasan terpendek dari
    a ke semua simpul lainnya}
8
9   KAMUS  s1,s2,...,sn  :  interger
    {larik interger}
10
11  d1,d2,...,dn  :  interger  {larik
    interger} i : interger
12
13  ALGORITMA
14  {Langkah 0 (inisialisasi) : }
15  for i ← 1 to n do
16        si ← 0
17        di ← mai
18  endfor
19
20  {Langkah 1: }
21        sa ← 1
22  {karena simpul a adalah simpul
    asal   lintasan terpendek, jadi
    terpilih      dalam      lintasan
    terpendek}
23        da ← infinity
24  {tidak  ada  lintasan  terpendek
    dari simpul a ke a}
25
26  {Langkah 2,3,...,n1 :}
27  for i ← 2 to n-1 do
    {Cari j sedemikian sehingga sj =
    0   dan dj = min (d1,d2,...,dn)}
28        Sj ← 1
29  {simpul j sudah terpilih ke dalam
    lintasan terpendek}
30  {perbarui   di,   untuk   i   =
31  1,2,3,...,n  dengan : di (baru) =
    min{di(lama),  dj + mji}
32  Endfor
```

2.2 *Floyd-Warshall Algorithm*

Algoritme ini ditemukan oleh Warshall untuk mencari lintasan terpendek adalah salah satu algoritme yang mudah di implementasikan karena termasuk algoritme yang sederhana. Masukan Algoritme ini merupakan matriks hubung graf berarah berlabel dan keluarannya merupakan lintasan

terpendek dari semua titik ke semua titik (Siang, 2011).

Algoritme ini juga merupakan salah satu algoritma routing protokol yang termasuk dalam *all-pairs shortest path* yang digunakan untuk mencari lintasan terpendek antara semua pasangan simpul pada graf berarah. Algoritme ini menggunakan matriks bobot n x n sebagai masukan, dimana n merupakan jumlah dari node. Algortima ini dapat mentolerir negative edge tetapi tidak untuk negative cycle. (Firdaus, et al., 2018).

Pseudocode algoritme *Floyd-Warshall*

```
1   let dist be a |V| × |V| array of
    minimum distances initialized to
    ∞ (infinity)
2   for each edge (u,v)
3       dist[u][v] ← w(u,v)  // the
    weight of the edge (u,v)
4   for each vertex v
5       dist[v][v] ← 0
6   for k from 1 to |V|
7       for i from 1 to |V|
8           for j from 1 to |V|
9               if dist[i][j] >
    dist[i][k] + dist[k][j]
10                  dist[i][j] ←
    dist[i][k] + dist[k][j]
11              end if
```

2.3 Link Failure

Link failure merupakan suatu masalah yang terjadi disaat melakukan proses routing terjadi suatu kegagalan link, jika jalur yang menghubungkan antara switch pada jaringan terjadi *link failure* maka akan mempengaruhi proses pertukaran data dan jalur tersebut tidak akan dapat dilewati. *Link failure* terjadi disaat jalur yang menghubungkan antara switch dengan switch lainnya terjadi down, saat terjadi down pada jalur yang menghubungkan itu maka routing protokol akan langsung melakukan update pada jalur lain yang dapat digunakan dengan tetap memilih jalur yang terpendek lainnya (Aprilianingsih, et al., 2017).

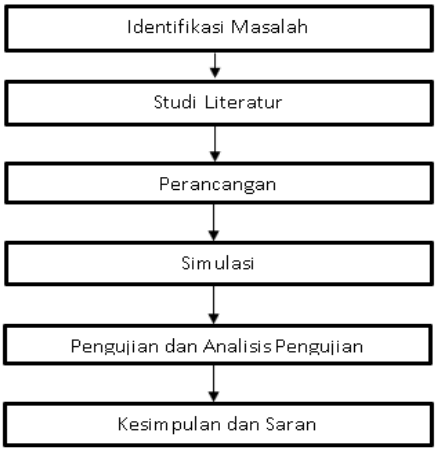
2.4 Convergence

Pengujian *Convergence* dilakukan untuk mengetahui waktu yang dibutuhkan sebuah jaringan untuk mencapai keadaan *steady state.* Apabila waktu yang dibutuhkan semakin kecil maka algoritma routing yang digunakan pada sistem dapat menemukan rute dengan cepat. (Firdaus, et al., 2018).

Nilai dalam *Convergance* dapat diketahui jika terdapat perubahan jaringan. Dalam menguji *Convergance* didapatkan waktu dengan rata-rata nilai detik. Tujuan dilakukanya pengujian *Convergence* adalah untuk mengukur seberapa cepat sebuah jaringan mendapatkan jaringan lain untuk menngantikan jalur sebelumnya.

3. METODOLOGI

Bab ini menjelaskan tentang langkah-langkah yang digunakan dalam peneliian, yaitu indetifikasi masalah, studi literatur, perancangan, simulasi, pengujian dan analisis, kesimpulan dan saran. Gambar 2 Menjelaskan tentang runtutan pengerjaan penelitian.

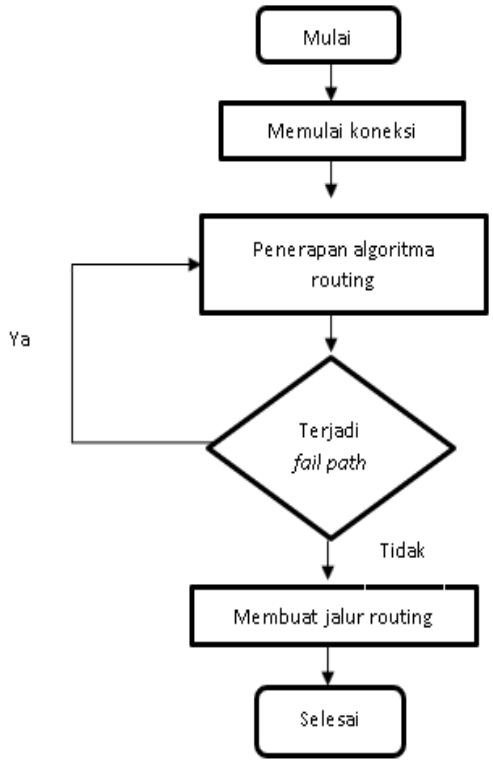


Gambar 1 Diagram alir penelitian

4. PERANCANGAN DAN IMPLEMENTASI

4.1 Alur Kerja Sistem

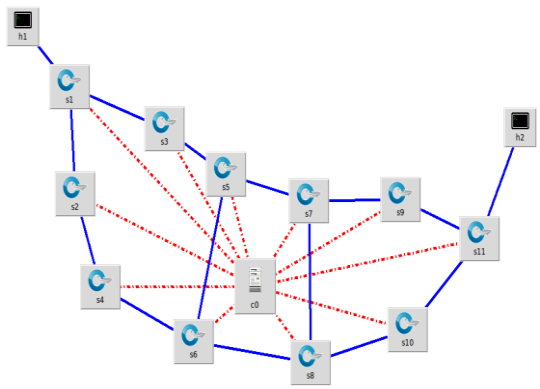
Cara kerja sistem atau alur kerja sistem saat kondisi normal maupun saat terjadi *link failure* pada penelitian ini akan digambarkan pada Gambar 3 Ryu controller akan digunakan sebagai controller setting pada sistem yang dibuat. Pada saat melakukan koneksi maka yang pertama kali dilakukan adalah setting port 6633 untuk ryu controller. Selanjutnya dilakukan penerapan algoritma *Dijkstra* dan juga *Floyd-Warshall* sebagai algoritma routing yang digunakan untuk menghasilkan jalur terpendek. Tetapi saat terjadi *link failure* akan dilakukan *routing* ulang dengan menggunakan algoritma *Dijkstra* dan juga *Floyd-Warshall.*



Gambar 2 Tahapan Perancangan Sistem

4.2 Perancangan Topologi

Perancangan topologi ini dibuat untuk nantinya akan dilakukan proses pengujian dengan melakukan penerapan algoritma-algoritma yang digunakan, yaitu algoritma Dijkstra dan juga Floyd-Warshall. Topologi yang akan digunakan adalah topologi Abilene yang dirancang akan menggunakan 2 host serta 11 switch dan topologi ini akan di terapkan pada emulator atau simulator mininet. Diharapkan pada topologi ini di saat terjadi *link failure* dapat menghasilkan jalur alternative lain dengan menerapkan algoritma Dijkstra dan juga Floyd-Warshall. Adapun gambaran topologi yang akan dibangun nanti adalah sebagaI berikut.

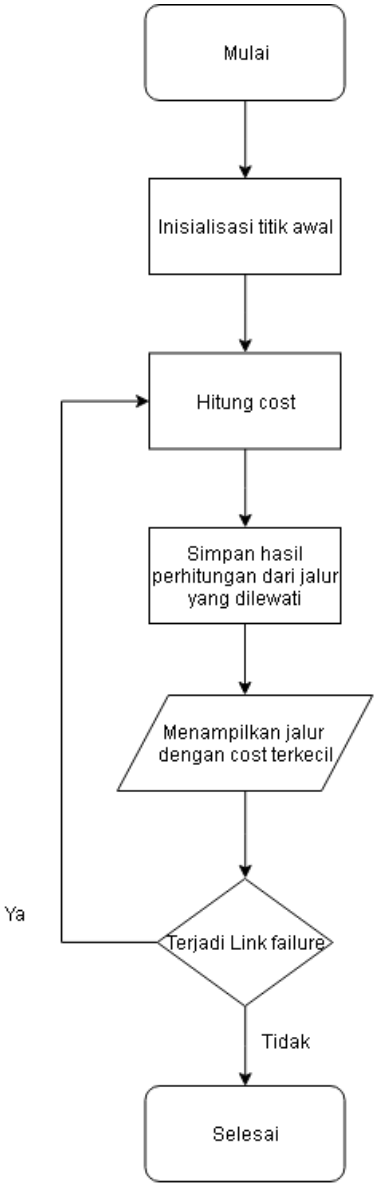


Gambar 3 Topologi *Abilene*

Gambar 4 dapat dilihat terdapat 2 host yaitu h1 dengan ip address 10.0.0.1, h2 dengan ip address 10.0.0.2. Lalu juga terdapat 11 switch dan 1 controller dengan 6633 sebagai port yang digunakan.

4.3 Perancangan Routing

Dalam perancangan routing akan menjelaskan tentang bagaimana membangun sebuah proses routing pada jaringan *openflow.* Pada proses pencarian jalur terpendek pada penelitian ini akan digunakan dua algoritma yaitu algoritma Dijkstra dan juga algoritma Floyd-Warshall. Dengan menerapkan algoritma Dijkstra dan juga Floyd-Warshall maka diharapkan akan dapat menemukan jalur terpendek mana yang akan dilewati dari node awal ke node tujuan.

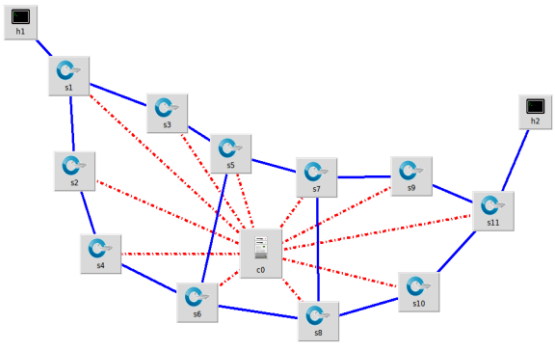


Gambar 4 Tahapan Perancangan Routing

Untuk menentukan jalur terpendek yang akan dilewati dari node awal ke node tujuan adalah dengan mempertimbangkan bobot terkecil sebagai parameter penentu yang digunakan. Dari menghitung setiap bobot yang telah dilewati dari setiap jalur lalu yang menghasilkan total bobot atau *cost* terkecil menjadi jalur yang akan dilewati. Proses dari penentuan jalur terpendek dapat dilihat pada gambar 5 dimulai dari menentukan titik awal atau node asal dengan melakukan inisialisasi pada topologi yang sudah dibuat, selanjutnya menghitung cost dari jalur yang telah dilewati, selanjutnya hasil yang sudah didapatkan disimpan sementara sampai menemukan jalur dengan cost terkecil, selanjutnya menampilkan jalur terpendek, jika terjadi *link failure* maka akan dilakukan pencarian jalur terpendek alternatif lain yang dapat digunakan.

4.4 Perancangan Pengujian

Pada perancangan pengujian akan menjelaskan tentang skenario-skenario pengujian yang akan dibuat. Skenario pengujian yang akan dibuat berdasarkan parameter pengujian dan juga topologi yang dipakai, pada penelitian ini topologi yang akan digunakan yaitu topologi Abilene dengan 2 host dan juga 11 switch. Pengujian akan dilakukan dengan melakukan pemutusan link (*link failure*) pada topologi yang ada pada simulator mininet, pada simulator ini memiliki fitur untuk membuat link yang dipilih dalam sebuah topologi menjadi down.



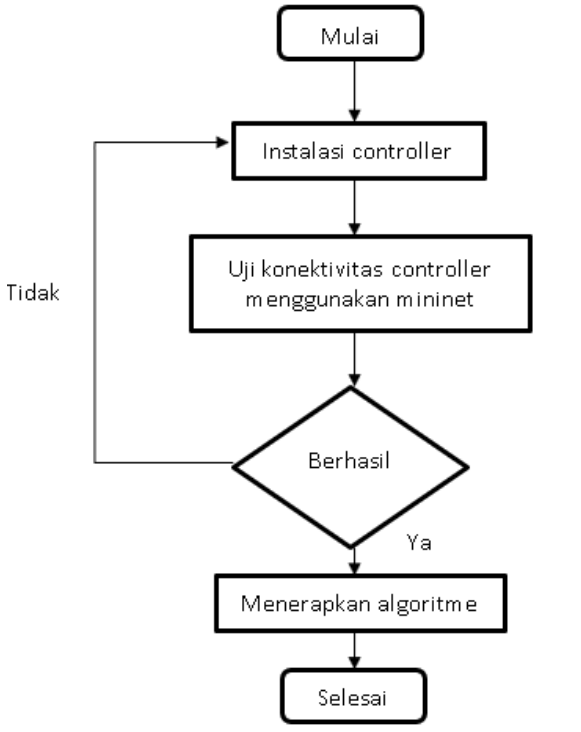
Gambar 5 Topologi *Abilene*

Pada gambar 6 dapat dilihat topologi Abilene, scenario pengujian yang dibuat akan berdasarkan topologi yang ada. Adapun scenario pengujian yang ada adalah sebagai berikut :

1. Tidak terjadi *link failure* pada saat pencarian jalur terpendek
2. Terjadi *link failure* antara h1 dengan h2 pada switch 7 dan switch 9 sehingga membutuhkan waktu convergance untuk melakukan routing ulang
3. Terjadi *link failure* antara h1 dengan h2 pada switch 5 dan switch 7 sehingga membutuhkan waktu convergance untuk melakukan routing ulang

4.5 Implementasi Sistem

Pada gambar 7 merupakan tahapan-tahapan yang dilakukan pada proses implementasi sistem. Dimulai dari melakukan instalasi *controller* yang digunakan, setelah instalasi *controller* telah dilakukan maka akan dilanjutkan ketahapan selanjutnya yaitu pengujian konektivitas *controller* dengan menggunakan simulator mininet untuk mengetahui apakah *controller* sudah berhasil berjalan atau belum.



Gambar 6 Tahapan Simulasi Sistem

Mininet adalah sebuah emulator yang dapat membuat sebuah jaringan virtual dan merupakan salah satu emulator yang cocok untuk melakukan pengembangan *software defined network.* Parameter keberhasilan instalasi *controller* adalah berhasil atau tidaknya *controller* dalam menghubungkan beberapa komponen yang ada dalam emulator mininet seperti *host* dan *switch.* Jika *controller* dapat menghubungkan komponen-komponen

tersebut maka proses instalasi berhasil dilakukan, dan jika controller tidak dapat menghubungkan komponen-komponen tersebut maka dalam proses instalasi *controller* masih terdapat kesalahan dan harus mengulang proses intalasi *controller*. Ketika *controller* sudah berjalan pada emulator mininet maka selanjutnya tahap yang akan dilakukan adalah menerapkan algoritma-algoritma routing yang digunakan pada penelitian ini untuk menentukan jalur terpendek.

5. PENGUJIAN DAN ANALISIS

6.1 Hasil Pengujian

1. Algoritme Djikstra

Tabel 1 hasil pengujian Convergence pada algoritme Dijkstra

Djikstra	Skenario 1	Skenario 2	Skenario 3
P 1	0ms	18.001ms	13.001ms
P 2	0ms	19ms	19ms
P3	0ms	16.005ms	18ms
P4	0ms	12ms	18.001ms
P5	0ms	17.002ms	18.001ms
Rata-Rata	0ms	16,401ms	17.201ms

Pada tabel 1 dapat diketahui hasil dari pengujian *convergence time* yang dilakukan. Dilakukan 3 skenario pengujian dimana pada setiap scenario pengujian dilakukan 5 kali percobaan pengambilan sampel untuk mendapat nilai rata-rata *convergence time* pada setiap scenario pengujian.

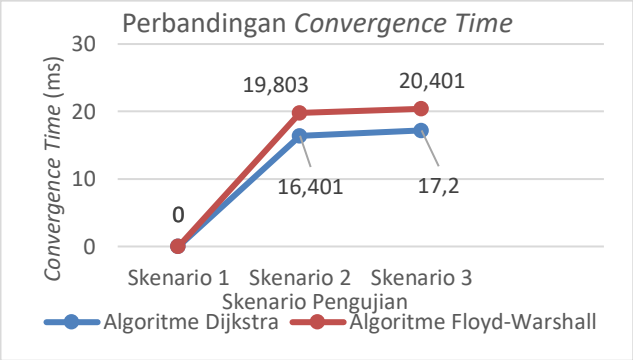
2. Algoritrme *Floyd-Warshall*

Tabel 2 hasil pengujian Convergence pada algoritme Floyd-Warshall

Floyd Warshall	Skenario 1	Skenario 2	Skenario 3
P 1	0ms	17.002ms	21.001ms
P 2	0ms	19.005ms	21.001ms
P3	0ms	20.005ms	19.002ms
P4	0ms	24.002ms	21.002ms
P5	0ms	19.001ms	20.001ms
Rata-Rata	0ms	19,803ms	20.401ms

Pada tabel 2 dapat diketahui hasil dari pengujian *convergence time* yang dilakukan. Dilakukan 3 skenario pengujian dimana pada setiap scenario pengujian dilakukan 5 kali percobaan pengambilan sampel untuk mendapat nilai rata-rata *convergence time* pada setiap scenario pengujian.

6.2 Analisis Pengujian



Gambar 7 Grafik Perbandingan Convergence Time

Dari hasil yang didapatkan pada gambar 8 dapat diketahui bahwa algoritme *Floyd-Warshall* memiliki waktu *convergence time* yang lebih besar dibandingkan dengan algoritme Dijkstra, besarnya *convergence time* pada algoritme *Floyd-Warshall* disebabkan karena algoritme dinamis ini selain melakukan perhitungan untuk mendapatkan jalur pada sebuah jaringan algoritme ini juga memperhitungkan konsekuensi dari setiap jalur yang diambil agar memiliki kemungkinan *error* yang mungkin terjadi saat proses *routing* dilakukan. Maka dari itu algoritme ini seperti melakukan dua kali perhitungan untuk mendapatkan sebuah jalur. Dari gambar grafik diatas juga dapat dilihat bahwa *trend* yang dimiliki grafik tersebut cenderung naik. *Trend* grafik yang cenderung naik ini disebabkan oleh beban pada proses pengujian yang dilakukan, karena pengujian yang dilakukan secara kontinuitas atau tidak berhenti dari skenario satu ke skenario selanjutnya memberikan beban pada proses perhitungan jalur tersebut.

6. KESIMPULAN

1. Sistem dapat melakukan pencarian jalur terpendek menggunakan Dijkstra algoritm dan juga Floyd-Warshall algorithm yang digunakan dalam keadaan normal maupun saat terjadi link failure.
2. Saat terjadi *link failure*, pada pengujian *convergence time* yang dilakukan menggunakan algoritme Dijkstra menghasilkan rata-rata waktu pada skenario 1 adalah 0ms karena pada skenario ini tidak terjadi *link failure* sehingga tidak membutuhkan waktu untuk menemukan jalur alternatif lain, pada skenario 2 menghasilkan 16.401 ms lalu pada skenario 3 menghasilkan 17.200 ms. Sedangkan pengujian

convergence time yang dilakukan menggunakan algoritme Floyd-Warshall menghasilkan rata-rata waktu pada scenario 1 adalah 0ms karena pada skenario ini tidak terjadi *link failure* sehingga tidak membutuhkan waktu untuk menemukan jalur alternative lain, pada scenario 2 menghasilkan 19.803 ms lalu pada skenario 3 menghasilkan 20.401 ms. Dari perbadingan waktu rata-rata tersebut algoritme Dijkstra memiliki hasil yang lebih baik dibandingkan dengan algoritme Floyd-Warshall.

7. SARAN

Pada penelitian ini terdapat beberapa saran yang dapat digunakan untuk penelitian selanjutnya. Pertama, jenis jaringan dapat diganti dengan objek lain karena algoritme Dijkstra dan juga algoritme Floyd-Warshall juga dapat diimplementasikan pada objek lain seperti untuk game. Kedua, algoritme Dijkstra dan juga algoritme Floyd-Warshall dapat diganti dengan algoritme *routing* lain karena untuk saat ini masih terdapat beberapa algoritme *routing* lain dan juga seiringnya perkembangan teknologi maka akan muncul algoritme *routing* baru yang juga dapat dibandingkan.

8. DAFTAR PUSTAKA

Aprilianingsih, E., Primananda, R., & Suharsono, A. Analisis Fail Path Pada Arsitektur Software Defined Network Menggunakan Dijkstra Algorithm. **Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer**, vol. 1, no. 3, p. 174-183, mei 2017. ISSN 2548-964X. Tersedia pada: <http://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/59>. [Diakses pada 14 Februari 2018]

Firdaus, M., Primananda, R., & Ichsan, M. Analisis Perbandingan Performansi Algoritme Floyd-Warshall dan Algoritme Johnson untuk Penentuan Rute Terpendek pada Software Defined Network. **Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer**, vol. 2, no. 9, p. 2469-2475, peb. 2018. ISSN 2548-964X. Tersedia pada: <http://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/2326>. [Diakses pada 20 Februari 2018]

Joestin, A. A., Najoan, M. E. & Manembu, P. D., 2015. Perancangan Routing Protocol Di Jaringan PT. Kawanua Internetindo. *E-Journal Teknik Elektro dan komputer,* Volume Vol.04 no.4. [online] Tersedia pada: <http://download.portalgaruda.org/article.php?article=332018&val=1028&title=Perancangan%20Routing%20Protocol%20Di%20Jaringan%20PT.%20Kawanua%20Internetindo> [Diakses pada 17 March 2018].

Musthofa, N., 2014. *Informatika*. [Online] Tersedia pada:<https://rizky-rafsamjani.blogspot.com/2014/11/standar-program-yang-baik.html>[Diakses pada 17 March 2018].

Pradnyana, I. N. P., 2010. Pencarian Rute Terpendek Tempat Penting Melalui Mobile GMaps dengan Menggunakan Algoritme Dijkstra. [Online] Tersedia pada:<http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2010-2011/Makalah2010/MakalahStrukdis2010-107.pdf> [Diakses pada 21 March 2018].

Siang, J. J., 2011. *Riset Operasi Dalam Pendekatan Algoritmis.* Yogyakarta: Penerbit Andi. [Online] Tersedia pada:<http://etheses.uin-malang.ac.id/6527/1/12650112.pdf> [Diakses pada 17 June 2018].

Waleed, S., Faizan, M., Iqbal, M. & Anis, M. I., 2017. Demonstration of Single Link Failure Recovery Using Bellman-Ford and Dijkstra Algorithm in SDN. [Online] Tersedia pada:< https://ieeexplore.ieee.org/document/7916533> [Diakses pada 3 September 2018].

Yunus, A., 2010. Implementasi Sistem Otentikasi Pada Pengguna Jaringan Hotspot Di Universitas Kanjuruhan Malang Guna Meningkatkan Keamanan

Jaringan Komputer. [Online] Tersedia di:<https://s3.amazonaws.com/academia.edu.documents/35131051/IMPLEMENTASI_SISTEM_OTENTIKASI_PENGGUNAN_HOTSPOT_DI_UNIV_KANJURUHAN.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1544534591&Signature=WXTWO6xBtBDNA7%2FgLGjZ0hZQi7A%3D&response-content-disposition=inline%3B%20filename%3DImplementasi_Sistem_Autentikasi.pdf> [Diakses pada 10 Desember 2018].