MENGATASI LINK FAILURE PADA SOFTWARE DEFINED NETWORK DENGAN ROUTING ULANG MENGGUNAKAN ALGORITMA DIJKSTRA

Bahrul Ulummudin, I Made Suartana[2]

[1,3] Teknik Informatika, Jurusan Teknik Informatika, Fakultas Teknik
Universitas Negeri Surabaya
[1]bahrul.17051204041@mhs.unesa.ac.id
[2]imadesuartana@unesa.ac.id

Abstrak— **Pengiriman data yang cepat menjadi tantangan di era pertumbuhan jaringan komputer yang sangat pesat ini. Pengiriman data dapat terganggu oleh beberapa faktor salah satunya *link failure*. *Single link failure* dapat terjadi rata-rata setiap 30 menit. Pada jaringan konvensional proses *routing* dan *forwarding* dilakukan pada satu perangkat, mengingat kondisi jaringan yang semakin berkembang dan semakin kompleks perlu adanya perkembangan pada arsitektur jaringan. SDN (*Software Defined Network*) menjadi salah satu paradigma baru yang muncul di bidang jaringan. SDN memisahkan antara *control plane* dan *data plane* sehingga tercipta lingkungan yang terpusat. pada penelitian ini skema jaringan SDN dibangun menggunakan *mininet* dengan menggunakan kontroler POX dengan menerapkan routing ulang (*rerouting*) menggunakan algoritma *Dijkstra* guna mengatasi *link failure* dengan mencari jalur terpendek lain setelah pada jalur utama terjadi link failure pada tiga desain topologi dengan jumlah *switch* dan *link* yang berbeda. Pada pengujian topologi algoritma *Dijkstra* yang di implementasikan pada POX *Controller* berhasil menemukan jalur baru pada saat terjadi *link failure* dalam jaringan. Pada pengujian waktu konvergensi di setiap topologi waktu konvergensi pada skenario kedua lebih baik dari pada skenario pertama, waktu konvergensi menjadi lebih sedikit seiring dengan kemungkinan jalur yang ada. *Latency* pada masing masing topologi juga memiliki kesamaan dimana *latency* paling sedikit ditunjukan oleh skenario pertama dan paling banyak ditunjukan oleh skenario kedua.**

***Kata Kunci*—Rerouting, SDN, Dijkstra, Mininet, POX .**

I. PENDAHULUAN

Pesatnya pertumbuhan jaringan komputer seiring dengan meningkatnya jumlah perangkat yang terhubung ke jaringan yang menyebabkan jaringan menjadi sangat kompleks. Kondisi ini menjadi tantangan untuk memberikan pelayanan pengiriman data yang lebih cepat ke tujuan. Pengiriman data dapat terganggu oleh beberapa faktor salah satunya *link failure* [2][3]. Secara khusus, *link failure* adalah masalah utama yang harus diatasi, di mana *single link failure* dapat terjadi rata-rata setiap 30 menit [1]. *Link failure* ini berdampak *packet loss* yang akan menyebabkan pengurangan *troughput* pada jaringan [4].

SDN (Software Defined Network) menjadi salah satu paradigma baru yang muncul di bidang jaringan. SDN memisahkan antara *control plane* dan *data plane* sehingga tercipta lingkungan yang terpusat. *Control plane* berguna untuk mengatur logika didalam perangkat jaringan misalnya routing table dan pemetaan jaringan. Sedangkan *data plane* berguna untuk meneruskan paket-paket yang masuk ke port pada perangkat jaringan menuju port keluar dengan instruksi dari *control plane*. Protokol openflow digunakan untuk komunikasi antara control plane dan data plane. *Software defined network* mempunyai beberapa kemampuan teknologi seperti *load balancing, multimedia multicast, Intrusion detection,* dan *routing* [5].

Routing digunakan untuk menunjang proses pengiriman data. Algoritma *routing* untuk menentukan jalur terpendek di antaranya algoritma *Dijkstra*. Algoritma *Dijkstra* merupakan algoritma rakus (greedy). Rakus disini merupakan cara algoritma *Dijkstra* dalam menemukan jalur terpendek dengan melakukan pengecekan pada setiap jalur sampai menghasilkan jalur dengan bobot yang paling kecil [5][6].

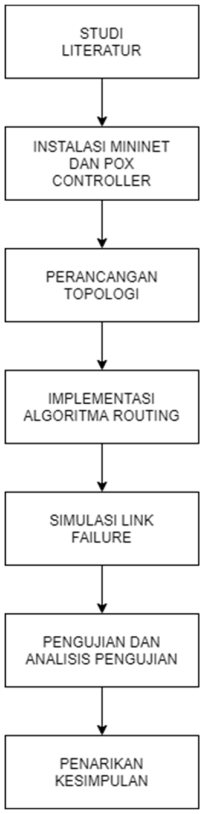
Risailin Dwi Jaka Fauzi, Rakhmadhany Primananda, dan Widhi Yahya melakukan pengujian Analisa perbandingan routing ulang dengan menggunakan algoritma *Dijkstra* dan algoritma *Floyd Warshall* dalam mengatasi *link failure* dengan *convergence time* sebagai parameter pembandingnya. Dua algoritma ini diujikan dengan 1 topologi dan 3 skenario yang sama. Hasilnya, pada skenario pertama kedua algoritma ini menghasilkan waktu 0ms. Sedangkan di skenario kedua algoritma *Dijkstra* menghasilkan 16.401 ms, dan algoritma *Floyd Warshall* menghasilkan 19.803 ms. Lalu di skenario ketiga algoritma *Dijkstra* menghasilkan 17.200 ms, dan algoritma *Floyd Warshall* menghasilkan 20.41 ms. Dari perbandingan waktu tersebut algoritma *Dijkstra* memiliki hasil yang lebih baik dibandingkan algoritma *Floyd Warshall* [6]. Eka Putri Aprilianingsih, Rakhmadhany Primananda, dan Aswin Suharsono membahas tentang analisis *fail path* pada *Software Define Network* menggunakan algoritma *Dijkstra*. Implementasi menggunakan topologi *mesh* dan *controller pyetric*. Algoritma akan di uji dengan 3 skenario. Didalam skenario terdapat 4 pengujian, yaitu pengujian topologi, *throughput, latency* dan *convergence*. Pengujian topologi dilakukan untuk menghasilkan jalur terpendek. Skenario 1 menghasilkan *cost* 10, Skenario 2 menghasilkan *cost* 13, dan Skenario 3 menghasilkan *cost* 18. Pengujian *troughput* mengalami penurunan dibandingkan saat belum terjadi *fail*

path. Pada pengujian *latency* waktu yang dibutuhkan data untuk mencapai tujuan lebih sedikit Ketika terjadi *fail path* dibandingkan sebelum terjadi *fail path*. Untuk pengujian *convergence* waktu yang dibutuhkan untuk mencari jalur baru pada skenario 3 lebih banyak dari pada skenario 2 [5].

Penelitian ini berfokus pada penggunaan routing ulang (*rerouting*) dengan algoritma *Dijkstra* untuk mengatasi *link failure*. Penelitian ini diimplementasikan pada *mininet* dan menggunakan kontroler POX yang menggunakan Bahasa *phyton*. Dengan ini diharapkan dapat mengatasi masalah gangguan pada *link* yang berupa *link failure* pada sebuah jaringan.

II. METODE PENELITIAN

Studi literatur menjadi Langkah pertama alur penelitian dalam mengatasi *link failure* dengan routing ulang (*rerouting*) menggunakan algoritma *Dijkstra*. Pelaksanaan studi literatur berguna untuk mengumpulkan serta mempelajari bahan-bahan dan teori baik dari jurnal, buku, atau sumber referensi lainya yang mendukung dilakukanya penelitian ini. Selanjutnya tahapan yang harus dilalui adalah membangun sistem yang diteliti. Penelitian ini membutuhkan *software* emulator jaringan Bernama mininet yang dipasang pada PC dengan system operasi ubuntu. Selain mininet, hal lain yang perlu dipasang pada PC adalah kontroler jaringan SDN POX. Sistem yang telah dibangun kemudian dianalisis kinerja dan diambil kesimpulan berdasarkan hasil yang telah didapatkan. Alur penelitian dijelaskan pada gbr.1.



Gbr. 1 Alur Penelitian

Parameter pengujian didasarkan pada tiga hal yaitu uji topologi, uji waktu konvergensi, dan uji *latency* yang didapat melalui 5 pengujian dengan 3 skenario pengujian pada uji topologi dan latency, dan 2 skenario pengujian pada uji waktu konvergensi.

A. Perancangan Sistem

Proses perancangan sistem pada penelitian ini adalah sebagai berikut :

1) Perancangan Algoritma *Dijkstra* : Algoritma *Dijkstra* ini akan menjadi inti dari kontroler SDN dibangun menggunakan Bahasa pemograman *python*, dan di implementasikan dalam POX kontroler. Perancangan algoritma *Dijkstra* ini memanfaatkan konsep graf berarah.

TABEL I
DAFTAR NOTASI YANG DIGUNAKAN

Simbol	Deskripsi
r_s	*Router* sumber
r_d	*Router* tujuan
r_{ic}	*Router* yang berdekatan di dalam *graf*
r_{jc}	*Router* yang berdekatan di dalam *graf*
P_{min}	*Path* terpendek *Dijkstra* berdasarkan jumlah *hops*
P_{Dset}	Himpunan semua solusi berbasis *Dijkstra*

Secara umum setiap *graf* sederhana G = (V, E) terdiri dari satu set *vertex* (yaitu *node*),V, serta satu set *edge* (yaitu *link*), E, yang terhubung dari satu titik ke titik yang lain. Di sini sebuah path didefinisikan P, dari sumber ke tujuan, sebagai urutan simpul berurutan yang mewakili node atau *router* dalam jaringan. Jalur dimulai pada *router* sumber, r_s, dan diakhiri dengan *router* tujuan, r_d, dengan r_{ic} dan r_{jc} menjadi dua *router* yang berdekatan di sepanjang P [7].

$$P = (r_s, \ldots, r_{ic}, r_{jc}, \ldots, r_d) \qquad (1)$$

Semua set jalur yang mungkin $P_{rs,rd}$ antara *router* sumber r_s dan *router* tujuan r_d di definisikan sebagai berikut :

$$P_{rs,rd} = \{P \mid (first(P) = r_s) \wedge (last(P) = r_d)\} \qquad (2)$$

Istilah jalur *Longest-Shortest* digunakan sebagai jalur yang memiliki jumlah hop maksimum di antara kumpulan solusi yang dikembalikan dengan menerapkan algoritma *Dijkstra* ke topologi jaringan untuk menemukan jalur terpendek antara setiap dua node yang mungkin dalam jaringan itu [7]. Untuk menemukan jalur *Longest-Shortest* ini, perlu fungsi khusus LS sebagai berikut:

$$LS(P_{Dset}) = x, such\ that\ x \in P_{Dset}\ and \\ \forall y \in P_{Dset} : len(y) \leq len(x) \quad (3)$$

Jika ada lebih dari satu jalur *Longest-Shortest* di P_{Dset}, algoritma memilih satu secara acak. P_{Dset} sendiri mewakili himpunan semua solusi berbasis *Dijkstra* untuk beberapa topologi jaringan (V, E) :

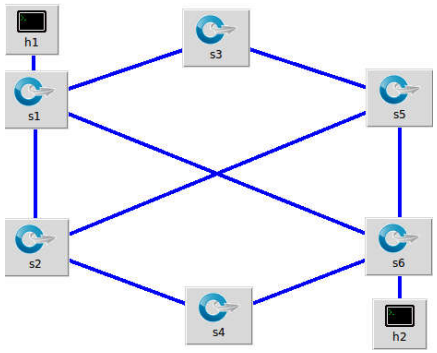
$$P_{Dset} = \{P \mid \forall\, r_s, r_d \in V : P = D(P_{rs,rd})\} \quad (4)$$

Dan jika ada *link failure* program otomatis menjalankan algoritma *Dijkstra* yang mencerminkan tindakan default yang

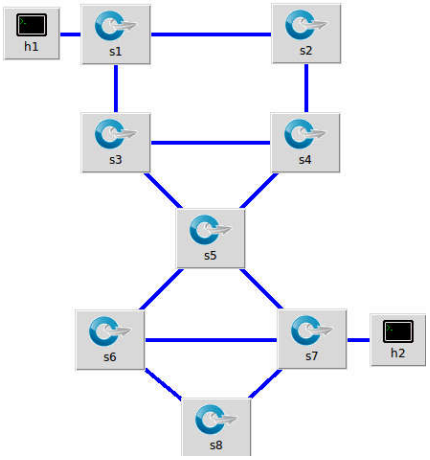
dilakukan oleh kontroler SDN pada saat-saat kegagalan, yaitu menghapus entri aliran dari jalur yang terpengaruh dan kemudian menginstal aturan cadangan dari *router* sumber r_s ke tujuan r_d, *pseudo code* ditunjukkan dalam (5).

$$\textbf{On Normal}: Set\ Primary\ Path\ as\ P_{min} \in P_{rs,rd}$$
$$\textbf{On Failure}: P_{rs,rd} := P_{rs,rd} - \{P_{min}\}$$
$$P_{min} := D(P_{rs,rd}) \qquad (5)$$

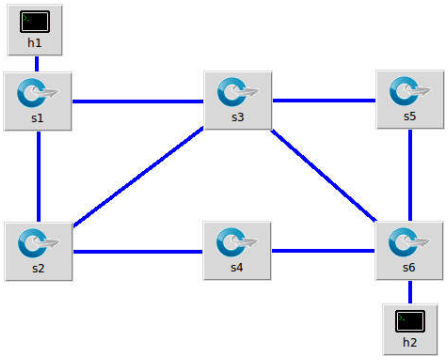
2) Perancangan Topologi: perancangan topologi dilakukan pada Mininet. Penelitian ini menggunakan tiga topologi dengan jumlah *switch* dan jalur yang berbeda. Desain topologi masing masing dapat dilihat di Gbr.2, Gbr.3, dan Gbr. 4



Gbr. 2 Topologi 1



Gbr. 3 Topologi 2



Gbr. 4 Topologi 3

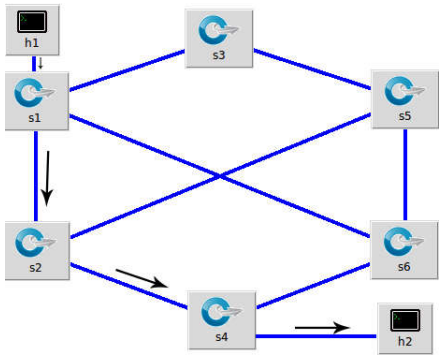
III. HASIL DAN PEMBAHASAN

Dalam pengujian ada beberapa skenario pengujian yang digunakan. Ada 3 skenario pengujian untuk pengujian topologi, dan pengujian *Latency*, dan ada 2 skenario pengujian untuk pengujian Waktu Konvergensi.

A. Pengujian Topologi

Pengujian ini bertujuan untuk mengetahui jalur terpendek yang harus dilalui untuk mengirimkan paket dari PC sumber ke PC tujuan. Pengujian ini dilakukan dengan melakukan *ping* dari h1 ke h2. Pengujian ini menggunakan dua skenario pengujian (dua kali pemutusan jalur routing).

Pada topologi 1 ditemukan jalur utama dari h1 ke h2 yakni S1-S2-S4, seperti pada Gbr.5.

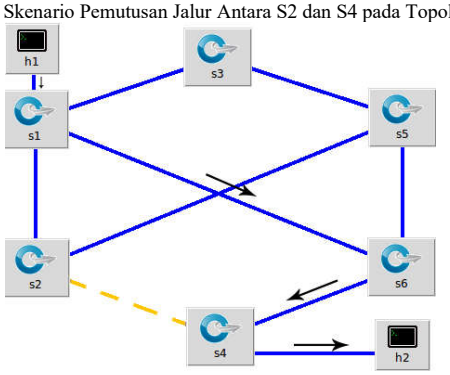


Gbr. 5 Jalur Pertama Topologi 1

Kemudian dilakukan pemutusan jalur pertama diantara S2 dan S4 dapat dilihat di Gbr. 6 dan Gbr. 7 yang menghasilkan jalur baru dari h1 ke h2 yaitu S1-S6-S4.

```
2 ---- 4  is removed
packet.src= 00:00:00:00:00:01  packet.dst= 00:00:00:00:00:02
src= 1  dst= 4
path= [1, 6, 4] current_p= [1, 6, 4]
```

Gbr. 6 Skenario Pemutusan Jalur Antara S2 dan S4 pada Topologi 1

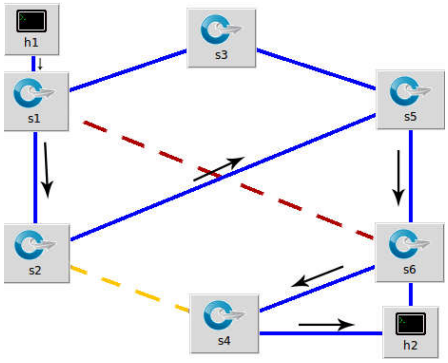


Gbr. 7 Jalur skenario 2 Topologi 1

Selanjutnya pemutusan jalur kedua dilakukan di antara S1 dan S6 dapat dilihat di Gbr. 8 dan Gbr. 9 dan menghasilkan jalur baru dari h1 ke h2 yaitu S1-S2-S5-S6-S4.

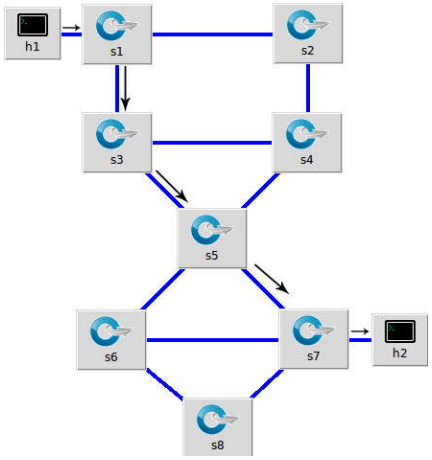
```
6 ---- 1  is removed
packet.src= 00:00:00:00:00:01  packet.dst= 00:00:00:00:00:02
src= 1  dst= 4
path= [1, 2, 5, 6, 4] current_p= [1, 2, 5, 6, 4]
```

Gbr. 8 Skenario Pemutusan Jalur Antara S1 dan S6 pada Topologi 1



Gbr. 9 Jalur Skenario 3 Topologi 1

Untuk topologi 2 memiliki jalur utama dari h1 ke h2 yakni S1-S3-S5-S7, seperti pada Gbr. 8.

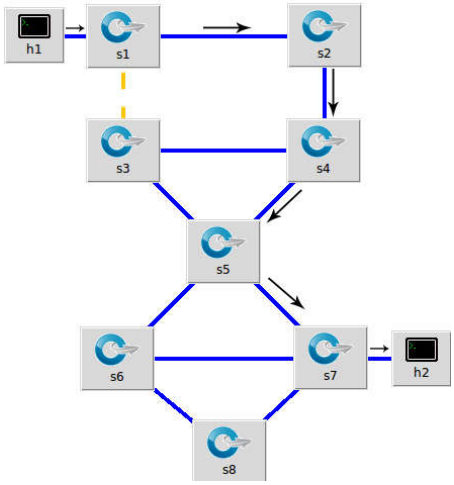


Gbr.10 Jalur Utama Topologi 2

Kemudian dilakukan pemutusan jalur pertama diantara S1 dan S3, dapat dilihat di Gbr. 11 dan Gbr. 12 yang menghasilkan jalur baru dari h1 ke h2 yaitu S1-S2-S4-S5-S7.

```
3 ---- 1  is removed
packet.src= 00:00:00:00:00:01  packet.dst= 00:00:00:00:00:02
src= 1  dst= 7
path= [1, 2, 4, 5, 7] current_p= [1, 2, 4, 5, 7]
```

Gbr. 11 Skenario Pemutusan Jalur Antara S1 dan S3 pada Topologi 2

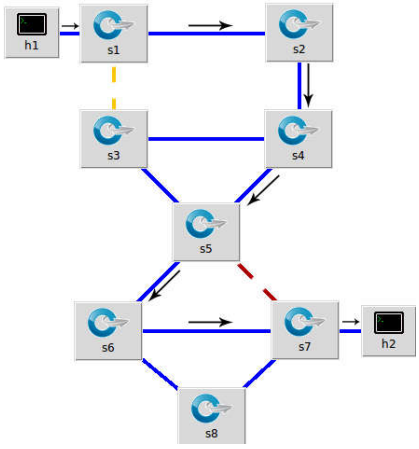


Gbr. 12 Jalur Skenario 2 Topologi 2

Selanjutnya pemutusan jalur kedua dilakukan di antara S5 dan S7, dapat dilihat di Gbr. 13 dan Gbr. 14 dan menghasilkan jalur baru dari h1 ke h2 yaitu S1-S2-S4-S5-S6-S7.

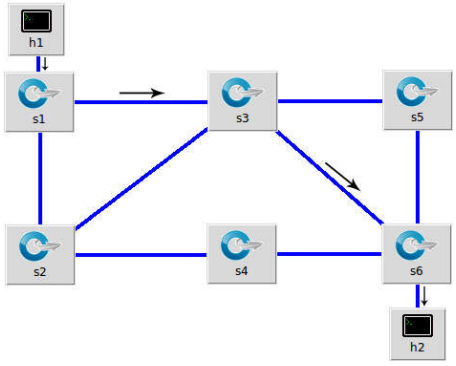
```
7 ---- 5  is removed
packet.src= 00:00:00:00:00:01  packet.dst= 00:00:00:00:00:02
src= 1  dst= 7
path= [1, 2, 4, 5, 6, 7] current_p= [1, 2, 4, 5, 6, 7]
```

Gbr. 13 Skenario Pemutusan Jalur Antara S5 dan S7 pada Topologi 2



Gbr. 14 Jalur Skenario 3 Topologi 2

Pada topologi 3 memiliki jalur utama dari h1 ke h2 yakni S1-S3-S6, seperti pada Gbr. 11.

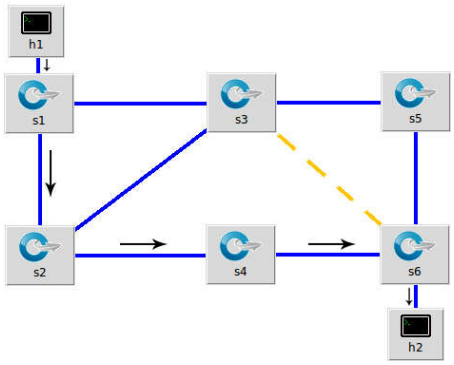


Gbr.15 Jalur Utama Topologi 3

Kemudian dilakukan pemutusan jalur pertama diantara S3 dan S6, dapat dilihat di Gbr. 16 dan Gbr. 17 yang menghasilkan jalur baru dari h1 ke h2 yaitu S1-S3-S5-S6.

```
3 ---- 6  is removed
packet.src= 00:00:00:00:00:01  packet.dst= 00:00:00:00:00:02
src= 1  dst= 6
path= [1, 2, 4, 6] current_p= [1, 2, 4, 6]
```

Gbr. 16 Skenario Pemutusan Jalur Antara S3 dan S6 pada Topologi 3

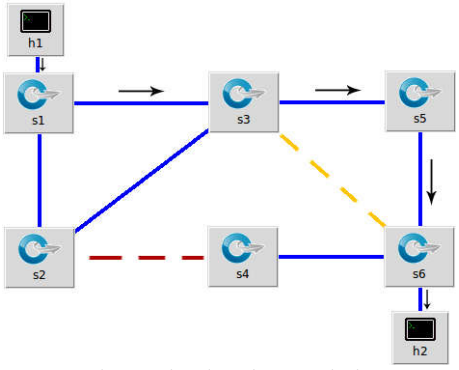


Gbr. 17 Jalur Skenario 2 Topologi 2

Selanjutnya pemutusan jalur kedua dilakukan di antara S2 dan S4, dapat dilihat di Gbr. 18 dan Gbr. 19 dan menghasilkan jalur baru dari h1 ke h2 yaitu S1-S3-S5-S6.

```
2 ---- 4  is removed
packet.src= 00:00:00:00:00:01  packet.dst= 00:00:00:00:00:02
src= 1  dst= 6
path= [1, 3, 5, 6] current_p= [1, 3, 5, 6]
```

Gbr. 18 Skenario Pemutusan Jalur Antara S2 dan S4 pada Topologi 3



Gbr. 19 Jalur Skenario 3 Topologi 2

B. *Pengujian Waktu Konvergensi*

Pengujian ini bertujuan untuk mengetahui waktu yang dibutuhkan kontroler untuk membentuk *table routing* baru. Pengujian waktu konvergensi ini menggunakan Teknik pemutusan jalur saat pengiriman paket *ping*, kemudian menghitung waktu konvergensi yang diperlukan agar jaringan bisa terhubung kembali. Untuk menghitung waktu konvergensi yaitu dengan cara melakukan perkalian antara jumlah *packet loss* dengan *timeout* seperti pada (6)[8].

$$Time\ Convergence = packet\ loss \times timeout \quad (6)$$

Pada topologi 1 skenario pengujian pertama saat h1 melakukan ping ke h2 terjadi pemutusan jalur diantara S2 dan S4, seperti pada Gbr. 7, kemudian dilakukan penghitungan seperti pada (6) dengan data yang diambil dari data ping h1 ke h2 seperti pada Gbr. 20.

```
root@ulum:~/mininet/custom# ping -s250 -c20 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 250(278) bytes of data.
258 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=7.03 ms
258 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=54.0 ms
258 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=36.3 ms
258 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=7.45 ms
258 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=113 ms
258 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=30.9 ms
258 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=14.7 ms
258 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=15.1 ms

--- 10.0.0.2 ping statistics ---
20 packets transmitted, 8 received, 60% packet loss, time 19301ms
rtt min/avg/max/mdev = 7.030/34.943/113.743/33.397 ms
```

Gbr. 20 Pengujian Waktu Konvergensi Topologi 1 Skenario 1

Dari data pada Gbr. 20 kemudian dihitung menggunakan (6). Dalam data tersebut diketahui *packet loss* berjumlah 60% dan *timeout* berjumlah 13 kemudian dilakukan penghitungan dengan mengalikan 60% dengan 13 yang menghasilkan angka 7,2 seperti pada Tabel II kolom pengujian ke 1 dan baris skenario 1. Pengujian seperti ini kemudian dilakukan berulang dengan masing masing skenario melakukan 5 pengujian.

Lalu dilakukan analisis hasil rata rata waktu konvergensi skenario 1 adalah 5,46 s, dan pada skenario pengujian 2 terjadi pemutusan jalur diantara S1 dan S6, seperti pada Gbr. 9, yang menghasilkan rata rata waktu konvergensi 3,69 s. Hasil pengujian lebih lengkap bisa dilihat pada Tabel II.

TABEL II
PENGUJIAN WAKTU KONVERGENSI TOPOLOGI 1

Jumlah pengujian	Skenario 1	Skenario 2
Pengujian ke 1	7,2	4,05
Pengujian ke 2	7,2	6,05
Pengujian ke 3	5	3,2
Pengujian ke 4	4,05	4,05
Pengujian ke 5	6,05	3,2
Rata - rata	5,9	4,11

Selanjutnya pada topologi 2 pengujian dilakukan sama dengan di topologi 1 yaitu dengan melakukan penghitungan (6) ke data yang diperoleh dari proses ping topologi kedua. Di skenario pertama saat h1 melakukan ping ke h2 dilakukan pemutusan jalur diantara S1 dan S3, seperti pada Gbr. 12, yang menghasilkan rata-rata waktu konvergensi 5,52 s, dan untuk skenario 2 pemutusan jalur terjadi diantara S5 dan S7, seperti Gbr. 14, yang menghasilkan rata rata waktu konvergensi 4,45 s. Hasil pengujian lebih lengkap bisa dilihat pada Tabel III

TABEL III
PENGUJIAN WAKTU KONVERGENSI TOPOLOGI 2

Jumlah pengujian	Skenario 1	Skenario 2
Pengujian ke 1	7,2	2,45
Pengujian ke 2	7,2	5
Pengujian ke 3	6,05	3,2
Pengujian ke 4	7,2	6,05
Pengujian ke 5	3,2	4,05
Rata - rata	6,17	4,15

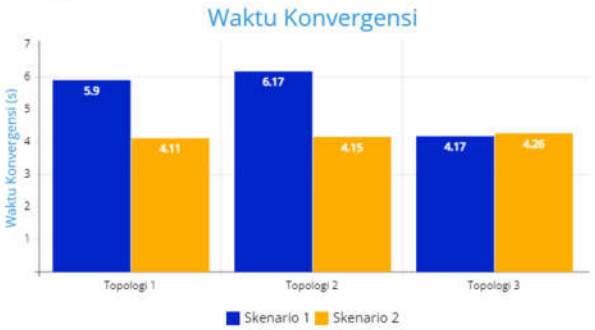
Kemudian di topologi 3 pengujian dilakukan sama dengan topologi 1 dan 2. Skenario pertama dilakukan pemutusan jalur diantara S3 dan S6, seperti pada Gbr. 17, saat h1 melakukan ping ke h2, yang menghasilkan waktu konvergensi rata rata 4.17 s, dan untuk skenario kedua saat h1 melakukan ping ke h2 dilakukan pemutusan jalur diantara S2 dan S4, seperti pada Gbr.

19, yang menghasilkan rata-rata waktu konvergensi 4,26 s. Hasil pengujian lebih lengkap bisa dilihat pada Tabel IV.

TABEL IV
PENGUJIAN WAKTU KONVERGENSI TOPOLOGI 3

Jumlah pengujian	Skenario 1	Skenario 2
Pengujian ke 1	1,8	4,05
Pengujian ke 2	5	5
Pengujian ke 3	4,05	5
Pengujian ke 4	5	4,05
Pengujian ke 5	5	3,2
Rata - rata	4,17	4,26

Perbandingan pengujian waktu konvergensi antara topologi 1,2, dan 3 dapat dilihat pada Gbr. 21.



Gbr. 21 Grafik Perbandingan Waktu Konfergensi

C. *Pengujian Latency*

Pengujian latency ini bertujuan untuk mengetahui waktu yang dibutuhkan paket untuk mencapai tujuan dengan menempuh jarak dari *node* asal ke *node* tujuan[5]. Pengujian *latency* ini dilakukan dengan mengirimkan paket *ping* dan mengamati RTT seperti pada Gbr. 22

```
root@ulum:~/mininet/custom# ping -s250 -c20 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 250(278) bytes of data.
258 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=52.5 ms
258 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=4.92 ms
258 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=6.12 ms
258 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=25.6 ms
258 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=11.1 ms
258 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=45.5 ms
258 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=15.3 ms
258 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=19.2 ms
258 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=42.2 ms
258 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=29.1 ms
258 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=11.1 ms
258 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=17.0 ms
258 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=5.27 ms
258 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=18.7 ms
258 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=36.2 ms
258 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=25.7 ms
258 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=5.24 ms
258 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=33.1 ms
258 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=47.8 ms
258 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=18.4 ms

--- 10.0.0.2 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19028m
rtt min/avg/max/mdev = 4.928/23.544/52.551/14.773 ms
root@ulum:~/mininet/custom#
```

Gbr 22 Pengujian Latency Topologi 1 Scenario 1

Dari data pada Gbr. 22 Ping dlakukan dengan mengirim paket sebesar 250 *byte* dan dilakukan sebanyak 20 kali. Dari data tersebut dapat dilihat pada RTT untuk menghitung *latency* bisa langsung dilihat pada *avg* atau *average* dari RTT tersebut. Dalam data tersebut ditemukan *latency* sebesar 23.544 ms seperti pada Tabel V kolom pengujian ke 1 dan baris skenario 1. Pengujian ini dilakukan berulang dengan 3 skenario dan 5 kali proses pengujian pada masing-masing skenario.

Pada topologi 1 rata rata *latency* dalam 5 kali percobaan. Pada skenario 1 pengujian dilakukan pada topologi tanpa ada *link failure* dan menghasilkan rata-rata *latency* 24.041 *ms*, pada skenario 2 pengujian dilakukan setelah terjadi *link failure* diantara S2 dan S4, seperti pada Gbr. 7, yang menghasilkan rata-rata *latency* 30.311 *ms*, kemudian di skenario 3 pengujian juga dilakukan pemotongan jalur diantara S1 dan S6, seperti pada Gbr. 9, dan menghasilkan rata-rata *latency* 29.163 *ms*. hasil pengujian *latency* topologi 1 lebih lengkap bisa dilihat pada Tabel V.

TABEL V
PENGUJIAN LATENCY TOPOLOGI 1

Jumlah pengujian	Skenario 1	Skenario 2	Skenario 3
Pengujian ke 1	23.544	27.311	27.725
Pengujian ke 2	28.012	21.005	28.406
Pengujian ke 3	21.525	26.520	24.219
Pengujian ke 4	21.480	29.104	26.361
Pengujian ke 5	25.455	29.048	23.083
Rata - rata	24.003	26.598	25.959

Selanjutnya pada topologi 2 di skenario 1 sama seperti skenario 1 topologi 1, tidak ada *link failure* dan menghasilkan rata rata *latency* 30.312 *ms*, pada skenario 2 pengujian dilakukan setelah terjadi *link failure* diantara S1 dan S3, seperti pada Gbr. 12, dan menghasilkan rata rata *latency* 30.238 *ms*, dan pada skenario 3 pengujian dilakukan setelah terjadi *link failure* antara S5 dan S7, seperti pada Gbr. 14, yang menghasilkan rata-rata *latency* 29.627 *ms*. hasil pengujian *latency* topologi 2 lebih lengkap bisa dilihat pada Tabel VI.

TABEL VI
PENGUJIAN LATENCY TOPOLOGI 2

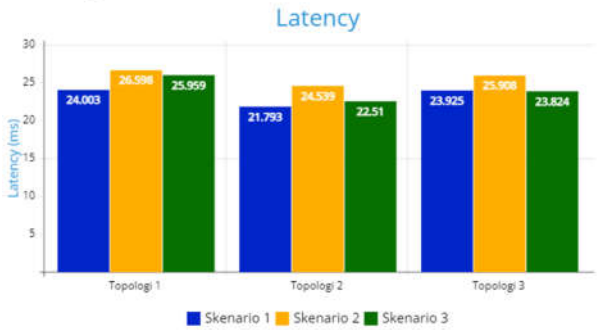
Jumlah pengujian	Skenario 1	Skenario 2	Skenario 3
Pengujian ke 1	25.048	23.874	23.696
Pengujian ke 2	25.032	17.106	23.932
Pengujian ke 3	17.247	28.081	26.077
Pengujian ke 4	13.699	21.149	21.618
Pengujian ke 5	27.940	32.484	17.228
Rata - rata	21.793	24.539	22.510

Kemudian di topologi 3 skenario 1 menghasilkan rata-rata *latency* 23.925 *ms*, di skenario 2 pengujian *latency* dilakukan setelah terjadi *link failure* diantara S3 dan S6, seperti pada Gbr. 17, yang menghasilkan rata-rata latency 25.908 *ms*, dan di skenario 3 pengujian *latency* dilakukan setelah terjadi *link failure* diantara S2 dan S4, seperti pada Gbr. 19 yang menghasilkan 23.824 *ms*. Hasil pengujian *latency* topologi 3 lebih lengkap bisa dilihat pada Tabel VII

TABEL VII
PENGUJIAN LATENCY TOPOLOGI 3

Jumlah pengujian	Skenario 1	Skenario 2	Skenario 3
Pengujian ke 1	21.249	25.048	25.577
Pengujian ke 2	25.259	27.334	21.427
Pengujian ke 3	27.527	27.635	26.067
Pengujian ke 4	23.428	25. 647	21.333
Pengujian ke 5	22.164	23.615	24.716
Rata - rata	23.925	25.908	23.824

Perbandingan pengujian *latency* antara topologi 1, 2, dan 3 bisa dilihat pada Gbr. 23.



Gbr. 23 Grafik Perbandingan Latency

IV. KESIMPULAN

Berdasarkan beberapa skenario pengujian terhadap algoritma *Dijkstra* yang digunakan sebagai algoritma routing ulang (*rerouting*) pada jaringan SDN, didapatkan hasil sebagai berikut :

1. Pada pengujian topologi, topologi pertama memiliki jalur utama S1-S2-S4, kemudian dilakukan simulasi *link failure* dengan memutus jalur diantara S2 dan S4, dan mendapatkan jalur S1-S6-S4. Setelah itu dilakukan simulasi *link failure* kedua dengan memotong jalur diantara S1 dan S6 yang menghasilkan jalur S1-S2-S5-S6-S4. Pada topologi kedua algoritma *Dijkstra* menghasilkan jalur utama S1-S3-S5-S7, kemudian dilakukan simulasi *link failure* pertama dengan memotong jalur diantara S1 dan S3 yang menghasilkan jalur baru S1-S2-S4-S5-S7, selanjutnya dilakukan simulasi *link failure* yang kedua dengan memotong jalur diantara S5 dan S7 dan menghasilkan jalur S1-S2-S4-S5-S6-S7. Untuk topologi ketiga algoritma *Dijkstra* menghasilkan jalur utama S1-S3-S6, kemudian dilakukan simulasi *link failure* pertama dengan memotong jalur diantara S3 dan S6 yang menghasilkan jalur baru S1-S2-S4-S6, selanjutnya dilakukan simulasi *link failure* yang kedua dengan memotong jalur diantara S2 dan S4 dan menghasilkan jalur S1-S3-S5-S6. Dengan ini dapat disimpulkan bahwa penerapan algoritma *Dijkstra* sebagai algoritma dalam routing ulang (*rerouting*) untuk menentukan jalur terpendek setelah terjadi *link failure* berhasil dilakukan dalam *software* emulasi jaringan *mininet*. Algoritma *Dijkstra* yang diterapkan dalam kontroler POX dapat menemukan jalur terpendek baru setelah terjadi link failure.
2. Ada beberapa perbedaan hasil pengujian terhadap penerapan algoritma *Dijkstra* di topologi 1, topologi 2, dan topologi 3 sebagai berikut : topologi 1 menghasilkan nilai rata-rata waktu konvergensi 5,46 s pada skenario pertama, dan 3,69 s pada skenario kedua. Selanjutnya pada topologi 2 nilai rata-rata waktu konvergensi adalah 5,52 s pada skenario pertama, dan 4,45 s pada skenario kedua. Kemudian pada topologi 3 nilai rata-rata waktu konvergensi adalah 4,17 s untuk scenario pertama, dan 4,26 s pada skenario kedua. Pada pengujian waktu konvergensi ini di setiap topologi waktu konvergensi di scenario kedua lebih baik dari pada waktu konvergensi di scenario pertama, sehingga dapat disimpulkan bahwa waktu konvergensi yang dibutuhkan lebih sedikit seiring dengan kemungkinan jalur yang ada karena algoritma *Dijkstra* memeriksa setiap jalur yang tersedia baru memilih jalur terpendek.
3. Dan untuk nilai rata-rata *latency* pada topologi pertama menghasilkan nilai 29.041 *ms* untuk skenario 1, 30.311 *ms* untuk skenario 2, dan 29.163 *ms* untuk skenario ketiga. Nilai rata-rata *latency* pada topologi 2 adalah 21.793 *ms* pada skenario pertama, 24.539 *ms* pada skenario kedua, dan 22.510 *ms* pada skenario ketiga. Kemudian di topologi 3 nilai rata-rata *latency* pada skenario pertama menghasilkan 23.925 *ms*, di skenario kedua menghasilkan 25.908 *ms*, dan 23.824 *ms* pada skenario ketiga.

UCAPAN TERIMAKASIH

Puji syukur penulis panjatkan kehadirat Allah SWT, karena atas segalanya yang telah diberikan penulis bisa menyelesaikan penelitian ini dengan baik. Tidak lupa juga penulis sampaikan terima kasih banyak kepada ibu karena selalu mendukung dan memberikan semangat. Terimakasih juga penulis sampaikan kepada dosen pembimbing karena atas bimbingan beliau penulis bisa menyelesaikan artikel ilmiah ini. Terakhir terimakasih untuk semua pihak yang selalu membantu penulis dalam penulisan artikel ini.

REFERENSI

[1] S. wang, H. Xu, L. Huang, X. Yang, dan J. Liu, “Fast Recovery for Single Link Failure with Segment Routing in SDNs”, IEEE 21st International Conference on High Performance Computing and Communications, 2019.

[2] Muthumanikandan V., dan Valliyammai C., “Link Failure Recovery Using Shortest Path Fast Rerouting Technique in SDN”, Wireless Pers Commun DOI 10.1007/s11277-017-4618-0, 2017.

[3] Rahmasari A.K.A., dan Suartana M.,“ Penerapan Algoritma Floyd-Warshall untuk Pemilihan Rute Routing OSPF pada Jaringan SDN”, JIEET: Volume 04 Nomor 01, 2020.

[4] Wibowo M.A., Yahya W., Kartikasari D.P.,“ Implementasi Link Fast-Failover Pada Multipath Routing Jaringan Software-Defined Network”, Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 2018.

[5] Aprilianingsih E.P., Primananda R., Suharsono A.,“ Analisis Fail Path Pada Arsitektur Software Defined Network Menggunakan Dijkstra Algorithm”, Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 2017.

[6] Fauzi R.D.J., Primananda R., Yahya W.,“ Perbandingan Routing Ulang Pada Algoritme Dijkstra dan Floyd-Warshall Dalam Mengatasi Link Failure Pada Arsitektur SDN”, Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 2019.

[7] Malik A., Aziz B., Ke C., Liu H., Adda M.,*" Virtual Topology Partitioning Towards an Efficient Failure Recovery of Software Defined Networks",* Converence : The 16th International Converence on Machine Learning and Cybernetics, 2017.

[8] Tersianto R.F., Hidayat N., Nurwasito H., *" Studi Komparasi Kinerja dari Adaptive Routing Protocol OSPFv3, RIPng, EIGRP IPv6, dan IS-IS pada Jaringan IPv6",* jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 2020.