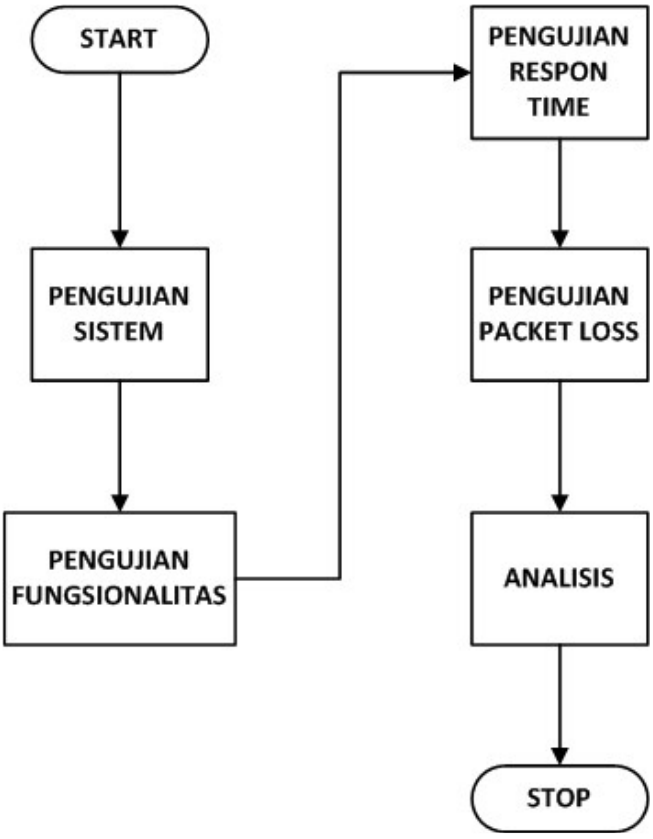
BAB 5 PENGUJIAN DAN ANALISIS

Pada bagian ini akan dijelaskan hasil pengujian tehadap implementasi yang telah dilakukan beserta analisisnya. Pengujian ini dilakukan untuk menilai seluruh kebutuhan fungsional maupun non-fungsional yang telah dispesifikasi sebelumnya. Analisis dilakukan untuk menilai apakah sistem bekerja sesuai yang diharapkan dan sebagai acuan untuk mendapatkan kesimpulan pada penelitian ini. Berikut alur pengujian dan analisis



Gambar 5. 1 Alur Pengujian dan Analisis

Berdasarkan gambar 5.1 menjelaskan alur pengujian dari sistem yang akan dilakukan dengan tiga pengujian yaitu pengujian fungsionalitas, pengujian response time dan pengujian *packet loss*. Setelah pengujian dilakukan akan diambil analisa tentang kinerja sistem ini.

5.1 Pengujian

5.1.1 Pengujian Fungsionalitas

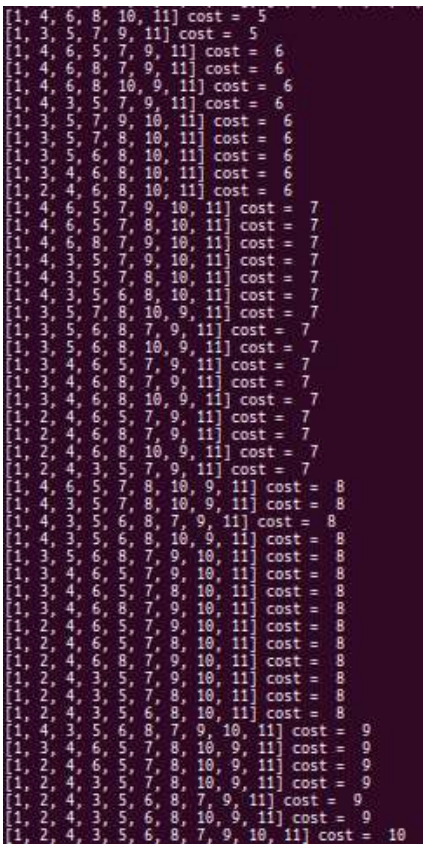
Pengujian fungsionalitas ini akan menjelaskan bagaimana mekanisme *fast-failover* dan *fail path (recovery)* di algoritme *Dijkstra* bekerja. Berikut pengujian *fast-failover* dan mekanisme *fail path (recovery).* Pada pengujian ini diemulasikan oleh emulator miniNAM. Fungsi dari emulator miniNAM yaitu dapat melihat

paket-paket yang dilewati. Untuk melihat paket ICMP yaitu menggunakan PING. Paket tersebut dapat terlihat pada emulator MiniNAM.

Pengujian ini melakukan beberapa skenario dari masing-masing algoritme. Dari masing-masing algoritme link yang diputus hanya ada 2 link. dari topologi terdapat 2 host dan 11 switch. Host 1 sebagai client dan host 2 sebagai server. Skenario 1 pada saat h1 ping ke h2 tidak ada pemutusan link pada sebuah topologi. Untuk skenario 2 jalur pengiriman paket yang pada skenario 1 akan diputus dan mengalihkan ke link yang lain. Sedangkan untuk skenario 3 jalur pengiriman paket pada skenario 2 akan diputus dan mengalihkan pada jalur yang lain.

5.1.1.1 Algoritme DFS dengan *fast-failover*

Pada pengujian ini dijelaskan bagaimana pengujian mekanisme *fast-failover* berjalan dengan menggunakan algoritme DFS. Skenario yang sudah ditentukan pada pengujian fungsionalitas sudah dijelaskan pada sub bab 3.5.1. Skenario tersebut untuk menentukan apakah sebuah *link* yang diputus dapat menemukan *link* yang baru lagi.

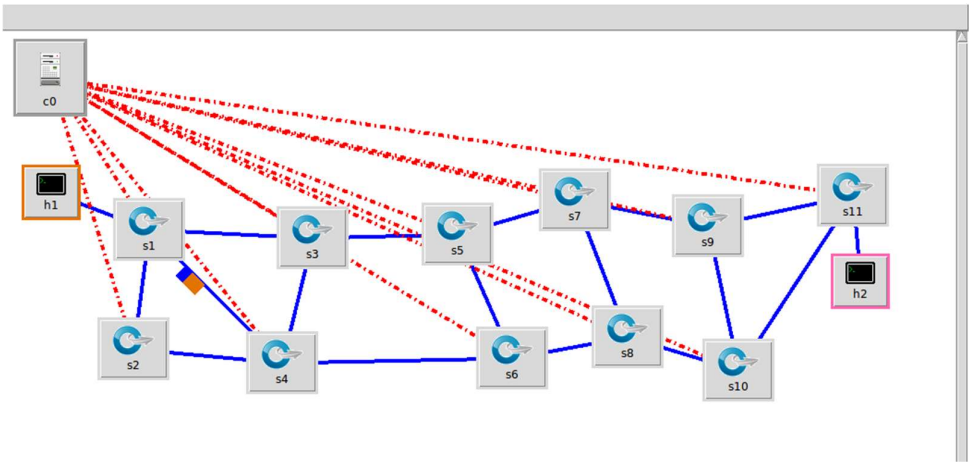


Gambar 5. 2 Tampilan beberapa jalur pada terminal *controller*

Pada gambar 5.2 menjelaskan ketika program dijalankan dan host 1 mengirimkan paket ICMP ke host 2, *controller* menampilkan jalur utama beserta jalur *backupnya* berdasarkan algoritme DFS.

Berikut beberapa skenario pada pegujian fungsionalitas dari algoritme DFS:

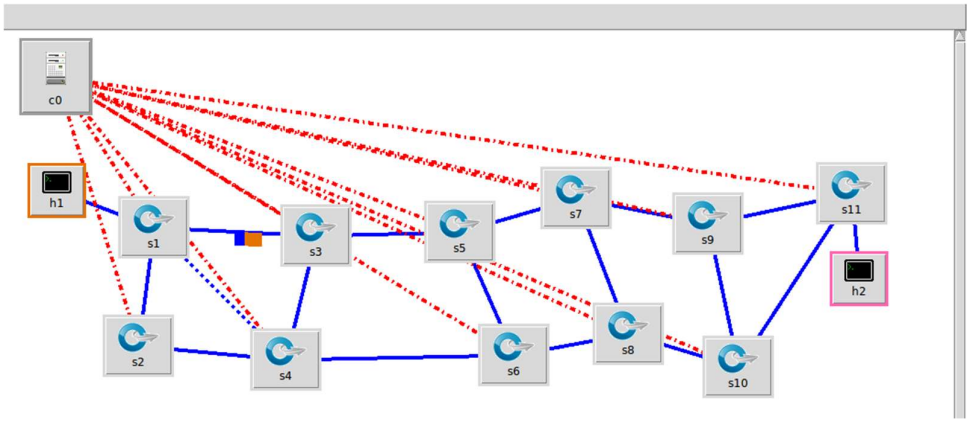
- Skenario 1



Gambar 5. 3 Topologi dengan tidak ada gangguan (DFS)

Pada gambar 5.3 menjelaskan ketika h1 melakukan ping ke h2. Pada gambar tersebut tidak ada kendala ketika h1 ke h2. paket ICMP yang ditunjukkan pada gambar tersebut melewati s1 ke s4. Secara keseluruhan paket yang dilewati yaitu s1-s4-s6-s8-s10-s11.

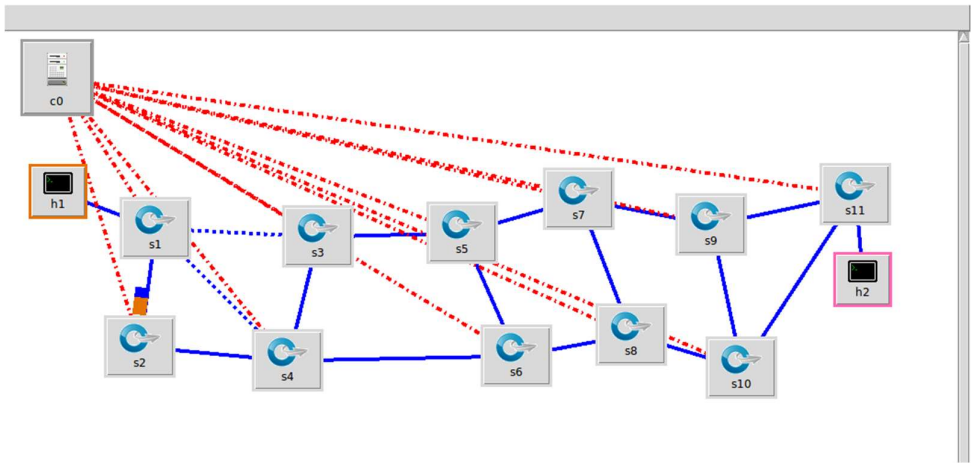
- Skenario 2



Gambar 5. 4 Topologi dengan link ke 1 diputus (DFS)

Berdasarkan Gambar 5.4 menunjukkan link ke 1 diputus yaiu link dari s1 ke s4 yang diputus, Sehingga jalur yang dialihkan s1 ke s3, jalur tersebut dapat ditunjukkan ketika paket ICMP melewati s1 ke s3. Secara keseluruhan dari asal ke tujuan yaitu paket melewati s1-s3-s5-s7-s9-s11.

- Skenario 3



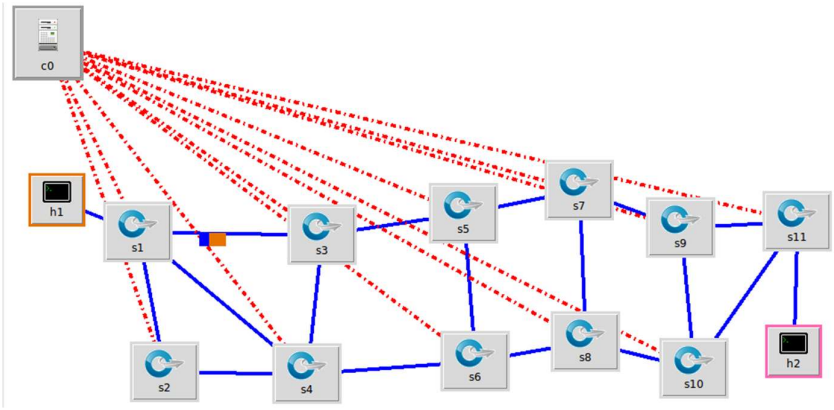
Gambar 5. 5 Topologi dengan link ke 2 diputus (DFS)

Berdasarkan gambar 5.5 menunjukkan link ke 2 diputus yaitu link s1 ke s3, sehingga link dialihkan pada alternatif link yang lain yaitu s1 ke s2. Berdasarkan penjelasan tersebut dapat dilihat paket ICMP yang melewati s1 ke s2. Secara keseluruhan dalam fungsionalitas pada skenario ini paket melawati beberapa switch yaitu s1-s2-s4-s6-s8-s10-s11.

5.1.1.2 Algoritme *Dijkstra* dengan Fail path (recovery)

Pada pengujian ini dijelaskan bagaimana pengujian *fail path (recovery)* yang ada pada algoritme Dijsktra. skenario yang sudah ditentukan pada pengujian fungsionalitas sudah dijelaskan pada sub bab 3.5.1. Skenario tersebut untuk menentukan apakah sebuah link yang diputus dapat menemukan link yang baru lagi. Berikut beberapa skenario pada pegujian fungsionalitas dari algoritme *Dijkstra:*

- Skenario 1

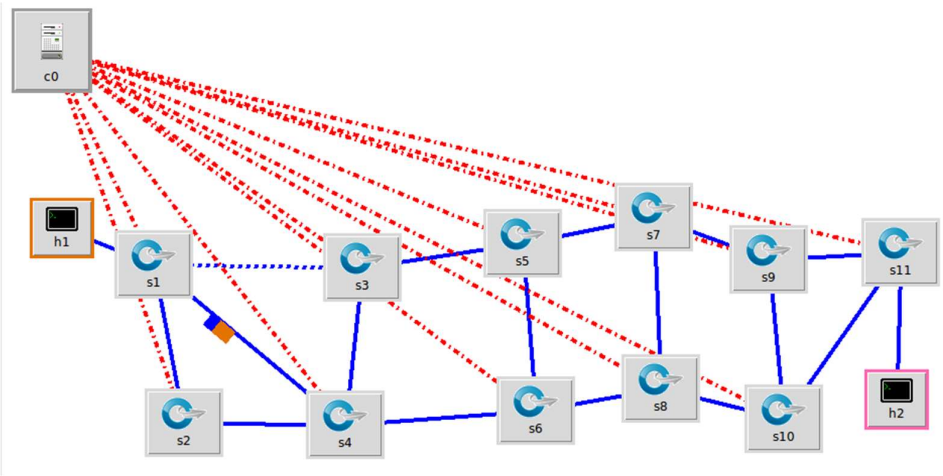


Gambar 5. 6 Topologi dengan tidak ada gangguan (*Dijkstra*)

Berdasarkan pada gambar 5.6 menunjukkan bahwa paket ICMP melewati s1 ke s3. Dari gambar tersebut belum ada gangguan link pada topologi.

Berdasarkan pencarian jalur algoritme Dijkstra Paket tersebut melewati beberapa switch, yaitu s1-s3-s5-s7-s9-s11.

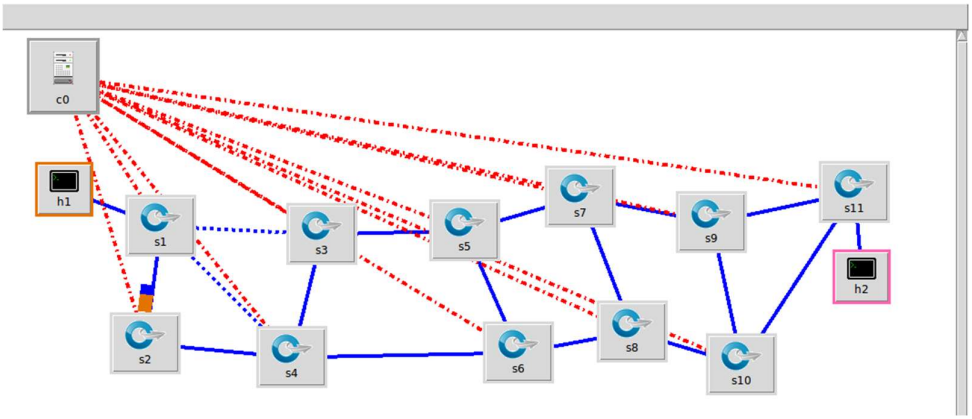
- Skenario 2



Gambar 5. 7 Topologi dengan link ke 1 diputus (*Dijkstra*)

Berdasarkan Gambar 5.7 menunjukkan link ke 1 diputus yaiu link dari s1 ke s3 yang diputus, Sehingga *link* yang dialihkan s1 ke s4, jalur tersebut dapat ditunjukkan ketika paket ICMP melewati s1 ke s3. Berdasarkan pada pencarian jalur algoritme *Dijkstra* switch yang kunjungi paket ICMP yaitu s1-s4-s6-s8-s10-s11.

- Skenario 3



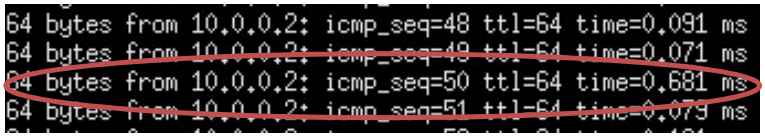
Gambar 5. 8 Topologi dengan link ke 2 diputus (*Dijkstra*)

Berdasarkan gambar 5.8 menunjukkan link ke 2 diputus yaiu link s1 ke s4 yang diputus, Sehingga *link* yang dialihkan pada *switch* s1 ke s2, jalur tersebut dapat ditunjukkan ketika paket ICMP melewati s1 ke s4. Berdasarkan pada pencarian jalur algoritme *Dijkstra* switch yang kunjungi paket ICMP yaitu s1-s2-s4-s6-s8-s10-s11.

5.1.2 Pengujian Response Time

Pengujian *response time* adalah pengujian yang melihat respon waktu ketika *link* di *down* atau diputus dan menemukan jalur yang baru untuk meneruskan paket ke tujuan. Pengujian ini membandingkan fast failover pada pencarian jalur DFS dan *fail path (recovery)* pada algoritme *Dijkstra*.

Pada pengujian ini dilakukan dengan topologi yang meliputi 11 switch dan 2 host. Host 1 sebagai clien dan host 2 sebagai server. Untuk mengukurt *response time* pada penelitian ini menggunakan *tool ping* diantara h1 dan h2. Penggunaan ping tersebut dikarenakan terdapat paket ICMP yang digunakan untuk menentukan tujuan dapat dijangkau dan berapa lama paket yang dikirimkan. Maka dari itu pengujian *response time* diambil waktu dari *ping* yang menentukan waktu ketika jalur dialihkan.



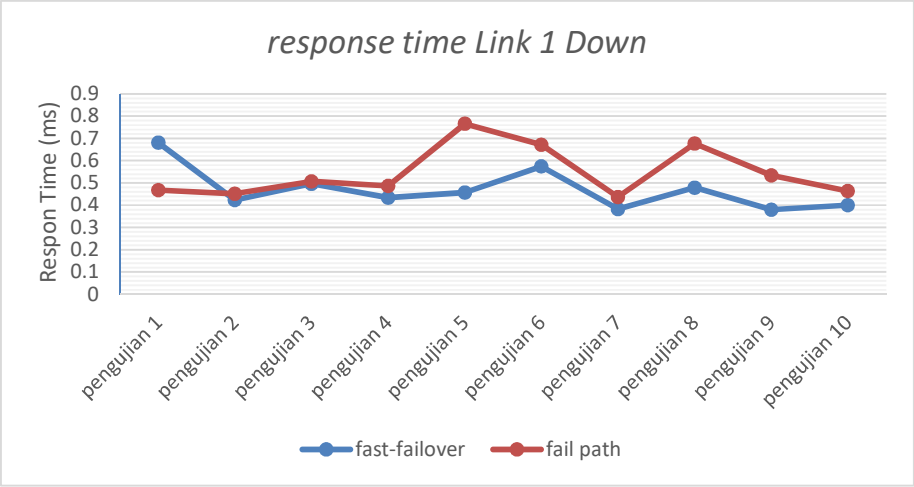
Gambar 5. 9 Contoh ping ketika *link* di down

pada gambar 5.9 dilihat response time ketika *link* down membutuhkann waktu 0.681 ms untuk menemukan jalur backup. Pengujian ini dilakukan sebanyak 10 kali dan pada topologi akan diputuskan 2 *link* secara bergantian dengan menggunakan masing – masing pencarian jalur pada penelitian ini, dapat dilihat pada tabel 5.1 sebagai berikut:

Tabel 5. 1 Tabel pengujian *response time*

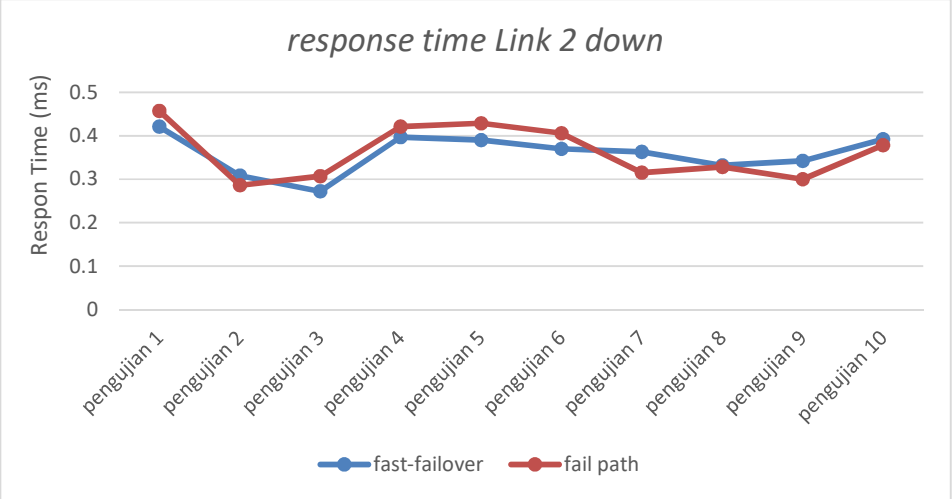
Pengujian ke-	*Fast-failover*		*Fail path (recovery)*	
	Link 1 down (ms)	*Link* 2 down (ms)	*Link* 1 down (ms)	*Link* 2 down (ms)
1	0.681	0.421	0.468	0.457
2	0.423	0.308	0.452	0.286
3	0.496	0.272	0.507	0.307
4	0.434	0.397	0.487	0.421
5	0.457	0.39	0.766	0.429
6	0.574	0.37	0.672	0.406
7	0.383	0.363	0.437	0.315
8	0.479	0.332	0.678	0.328
9	0.38	0.342	0.534	0.3
10	0.401	0.392	0.464	0.378
Rata-rata	0.471	0.359	0.547	0.363

Dari pengujian *response time* tersebut diambil data dari 10 kali pengujian. Data tersebut dimasukkan pada sebuah table dapat dilihat pada tabel 5.1. tabel tersebut mengambil percobaan dengan memutuskan 2 *link* pada topologi yang digunakan untuk penelitian ini. Berikut bentuk grafik dari pengujian response time ketika link ke 1 diputus.



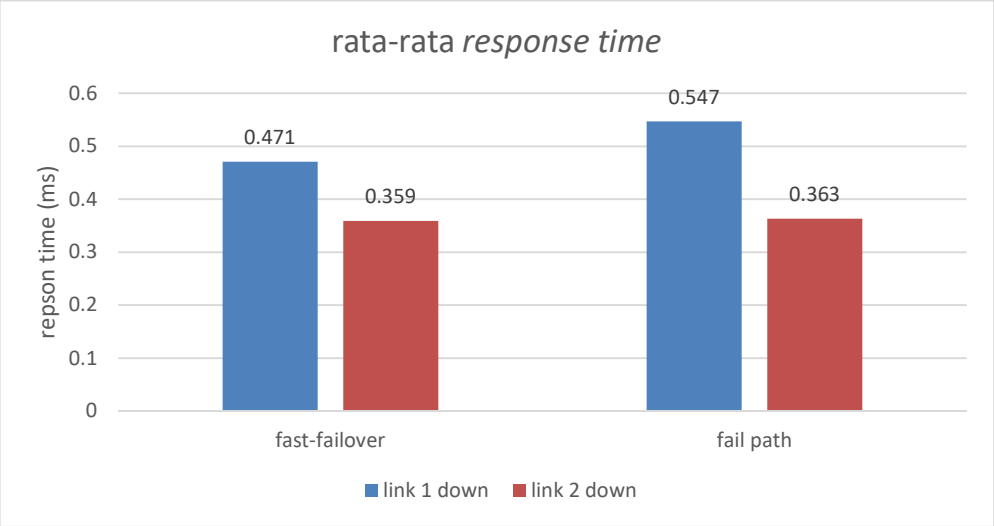
Gambar 5. 10 Response time ketika *link* ke-1 di down

Berdasarkan gambar 5.10 Grafik membandingkan ketika *link* ke 1 di putus diantara mekanisme *fast-failover* dan mekanisme fail path (recovery). Pada grafik diatas merupakan perbandingan antara mekanisme *fast-failover* dengan mekanisme fail path (recovery). Untuk response time ketika link ke 2 diputus ditunjukkan pada gambar 5.11.



Gambar 5. 11 Response time ketika link ke 2 down

Berdasarkan Gambar 5.11 dijelaskan ketika kedua *link* tersebut diputus atau *link* yang kedua diputus. *Response Time* tersebut memiliki beberapa pengujian yang tidak jauh beda. Di grafik diatas lebih unggul mekanisme *fast-failover* daripada mekanisme *fail path (recovery)*.



Gambar 5. 12 Grafik perbandingan response time *fast-failover* dan *fail path (recovery)*

Dari 2 pengujian *Response time* yang telah dilakukan *response time* pada pengujian 1 *link* diputus dengan mekanisme *fast-failover* lebih unggul daripada rata-rata *response time* pada mekanisme *fail path (recovery)*. Dan pada pemutusan *link* yang kedua *response timenya* tidak berbeda jauh tetapi masih lebih unggul yang dimiliki mekanisme *fast-failover.* Dari Rata-rata tersebut *mekanisme fast-failover* lebih unggul dikarenakan paket tidak kembali pada kontroller untuk meminta jalur baru tetapi terus berjalan dengan mengalihkan *backup link*. sebaliknya pada mekanisme *fail-path* dia meminta jalur baru lagi pada controller untuk meneruskan paket.

5.1.3 Pengujian *Packet Loss*

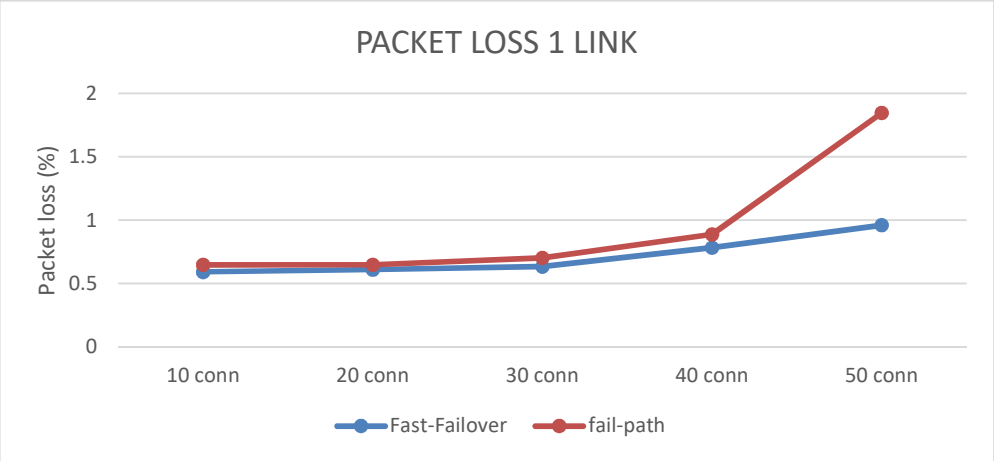
Pengujian *packet loss* adalah pengujian paket yang hilang ketika *link* di down pada saat mengirim paket sampai menemukan jalur yang baru. Pada pengujian *packet loss* ini membandingkan antara mekanisme *fast failover* menggunakan algoritme pencarian DFS dengan mekanisme *fail path (recovery)* yang ada di *Dijkstra.*

Untuk mengujikan *packet loss* pada penelitian ini menggunakan perintah *iperf* di masing-masing *host* pada topologi. Pada *host* 2 (h2) sebagai *server* menggunakan perintah *iperf -s -u* dan Host 1(h1) sebagai *client* dengan menggunakan perintah *iperf -c 10.0.0.2 -u -P 20 -t 50 -i 1.* Kedua *host* tersebut menggunakan protokol *UDP* untuk melihat *packet loss*. Pengujian ini melakukan percobaan dengan 10, 20, 30 40, dan 50 koneksi dan melakukan pengujian *packet loss* dengan 1 *link* di down, 2 *link* di down dan 3 *link* di down. Berikut tabel 5.2 hasil pengujian dari *packet loss*

Tabel 5. 2 Tabel pengujian *Packet loss*

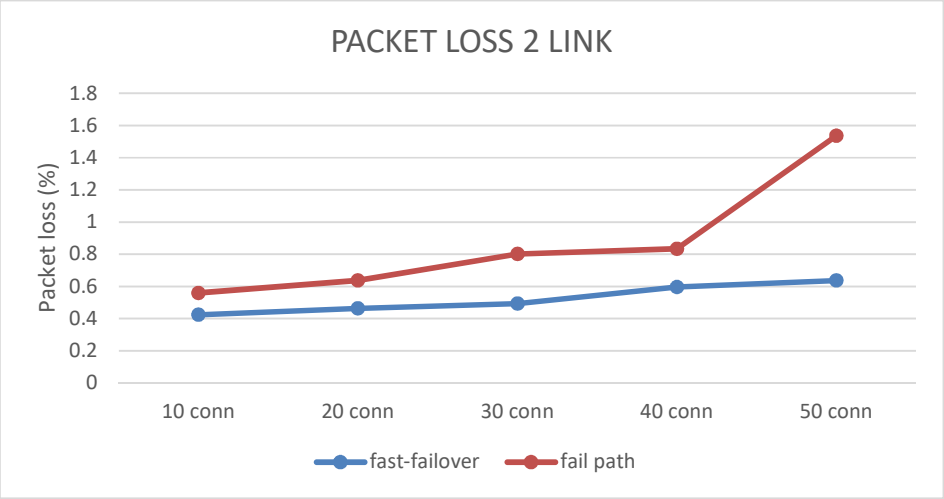
Jumlah koneksi	*Fast-failover* (%)			*Fail path (recovery)* (%)		
	1 *link* down	2 *link* down	3 *link* down	1 *link* down	2 *link* down	3 *link* down
10 conn	0.592	0.423	0.796	0.647	0.559	0.783
20 conn	0.61	0.4635	0.8775	0.6475	0.637	0.82
30 conn	0.634	0.493	0.9006	0.702	0.802	0.993
40 conn	0.78425	0.595	0.907	0.888	0.833	1.14
50 conn	0.9602	0.636	0.9382	1.846	1.536	1.88
Rata-rata	0.71609	0.5221	0.8839	0.9461	0.873	1.123

Nilai *packet loss* pada tabel diatas diambil berdasarkan skenario yang telah di ujikan. Nilai nilai tersebut berdasarkan jumlah koneksi yang terhubung dan d ambil beberapa *link* yang *down*. Pada tabel diatas yang diujikan hanyalah mekanisme *fast-failover* dengan mekanisme *fail path (recovery)*.



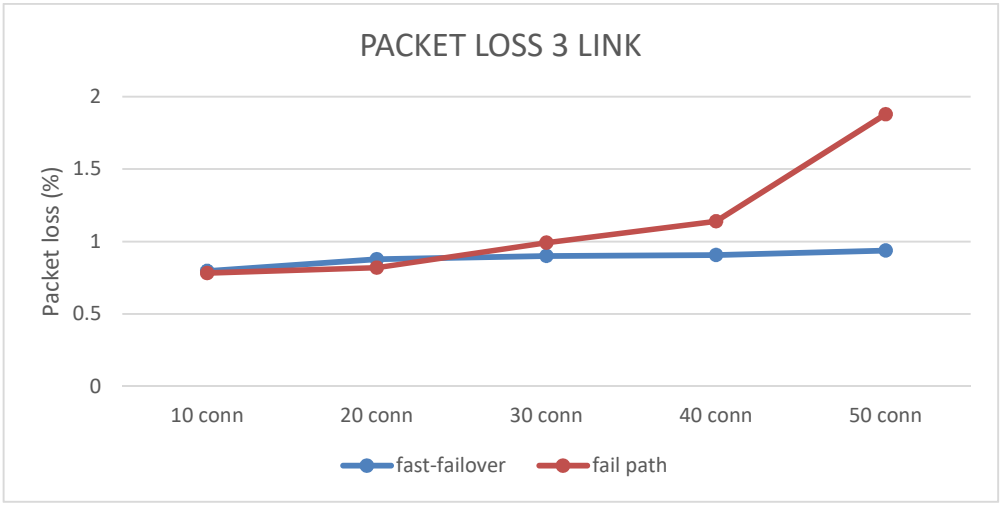
Gambar 5. 13 Grafik *Packet loss fast-failover* dan *fail path (recovery)* 1 *link down*

Berdasarkan gambar 5.13 menjelaskan *packet loss* ketika *link* yang diputus hanya 1 *link*. Berdasarkan dari grafik tersebut *packet loss* mekanisme fast-failover lebih rendah dibandingkan dengan mekanisme *fail path (recovery)*.



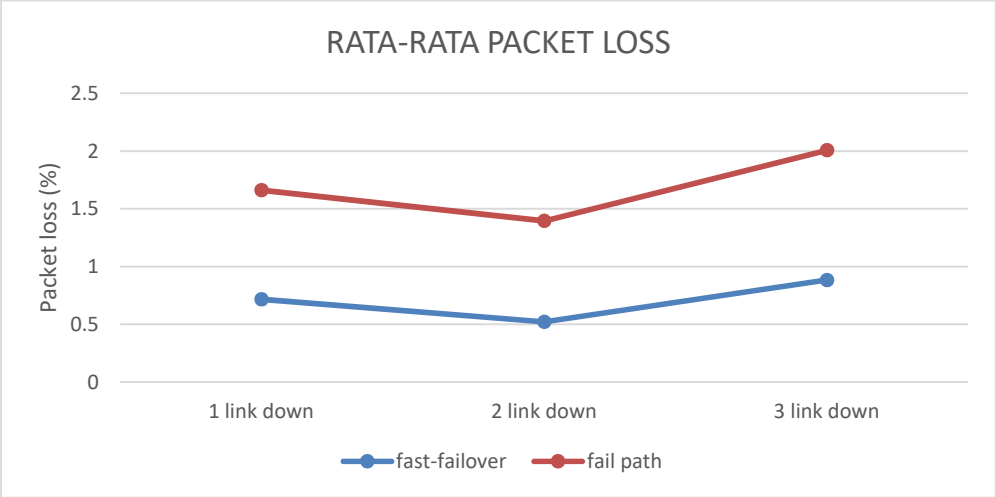
Gambar 5. 14 Grafik *Packet loss fast-failover* dan *fail path (recovery) 2 link down*

Berdasarkan gambar 5.14 dijelaskan *packet loss* yang *link* diputus ada 2 *link* yang diputus. Nilai *packet loss* tersebut mekanisme fast-failover lebih rendah dibandingkan dengan mekanisme *fail path (recovery)*. Packet loss mekanisme fast-failover stabil dapat dilihat pada grafik yang menunjukkan algoritme DFS.



Gambar 5. 15 Grafik *Packet Loss fast-failover* dan *fail path (recovery)* 3 *link down*

Pada gambar 5.15 menunjukkan ketika *link* yang diputus ada 3 *link*. Nilai pada grafik itu lebih stabil yang dimiliki mekanisme *fast-failover*.



Gambar 5. 16 Grafik rata-rata packet loss *fast-failover* dan *fail path (recovery)*

Setelah melakukan pengujian *packet loss* dengan beberapa pengujian diambil nilai rata-rata yang sudah dijelaskan pada sebuah grafik batang diatas. Dari packet loss tersebut rata-rata dari jumlah *link* yang diputus dan jumlah koneksi yang ditentukan mekanisme *fast-failover* lebih unggul dibandingkan dengan mekanisme *fail path (recovery)*.

5.2 Analisis

5.2.1 Analisis Fungsionalitas

Berdasarkan pada skenario pengujian fungsionalitas yang telah dilakukan, sistem berhasil melakukan pengalihan *link* pada topologi pengujian dengan menggunakan 2 algoritme pencarian jalur. Pada skenario pertama DFS dengan mekanisme *fast-failover* dapat menemukan jalur s1-s4-s6-s8-s10-s11 sedangkan pada algoritme *Dijkstra* dengan mekanisme *fail path (recovery)* menemukan jalur s1-s3-s5-s7-s9-s10-s11. Pada skenario pertama tidak ada perbedaan jumlah link yang ditemukan, namun yang berbeda adalah jalur yang ditemukan.

Skenario kedua pada saat pemutusan *link* dari pencarian jalur algoritma DFS pada skenario pertama link s1 ke s4 diputus dapat menemukan jalur yang lain yaitu s1-s3-s5-s7-s9-s11, sedangkan pada algoritme *Dijkstra link* yang diputus s1 ke s3 dapat menemukan jalur yang baru yaitu s1-s4-s6-s8-s10-s11. Pada skenario ini jumlah *switch* yang dikunjungi sama, namun jalur yang didapat setelah pemutusan link berbeda.

Skenario ketiga pemutusan *link* yang ditemukan pada kedua algoritma tersebut ketika *link* pada skenario kedua diputus, jalur baru yang ditemukan yaitu s-1-s2-s-4-s6-s8-s10-s11. Untuk skenario ini tidak jalur yang berbeda dan *switch* yang di kunjungi juga sama.

Dari hasil yang didapat dalam pengujian ini untuk skenario 1 dan skenario 2 berbeda, karena perbedaan tersebut menggunakan aturan masing-masing algoritme. Skenario dilakukan karena pada algoritme DFS dengan mekanisme

f*ast-failover* hanya dapat memutuskan 2 *link* dikarenakan pada saat h2 membalas paket tersebut paket tidak diterima pada h1.

5.2.2 Analisis *Response Time*

Berdasarkan pengujian *Response time* yang telah dilakukan dengan mengukur waktu setelah *link* diputus dan sampai menemukan *link* yang baru. Sistem melakukan pengujian sebanyak 10 kali pada masing-masing mekanisme. serta jalur yang diputus sesuai pada pengujian fungsionalitas penelitian ini.

Berdasarkan hasil pengujian ini, waktu untuk mengalihkan jalur ketika *link* yang diputus berdasarkan mekanisme *fast-failover* dengan pencarian jalur DFS dapat diketahui waktu yang dibutuhkan ada saat *link* ke 1 dan *link* ke 2 diputus. untuk *link* ke 1 sistem membutuhkan waktu 0.38-0681ms, sedangkan untuk *link* yang ke 2 sistem mebutuhkan waktu 0.272-0.421.

Hasil dari pengujian *response time* untuk mekanisme *fail path (recovery)* pada algoritme *Dijkstra* dapat diketahui waktu yang dibutuhkan untuk menemukan jalur yang baru pada asaat *link* ke 1 dan link ke 2 diputus. untuk *link* ke 1 sistem memerlukan waktu 0.437-0.766ms sedangkan *link* ke 2 membutuhkan waktu 0,286-0,457 ms.

Dari hasil yang didapatkan pada saat pengujian Pada pemutusan *link response time* pada mekanisme *Fast-failover* dengan pencarian jalur DFS lebih kecil dibandingkan *fail path (recovery)* yang ada pada algoritme *Dijkstra*. Karena mekanisme *fast-failover* tidak perlu memberitahu *controller* bila ada jalur yang diputus untuk menemukan jalur yang baru, sedangkan pada *fail path (recovery)* ketika *link* diputus sistem memberitahu controller jika ada *link* yang putus dan *controller* mencarikan *link* yang baru.

5.2.3 Analisis *Packet Loss*

Berdasarkan hasil pengujian *packet loss* yang telah dilakukan dengan melihat paket yang hilang pada saat pemutusan *link*. Pengujian ini melakukan percobaan dengan 10-50 koneksi dan melakukan 3 *link* yang diputus pada mekanisme *fast-failover* dan mekanisme *fail path (recovery)*. Dari hasil yang didapatkan, *packet loss* pada pengujian mekanisme *fast-failover* lebih kecil dibandingkan dengan mekanisme *fail path (recovery)*.

Pada pengujian *packet loss* pada mekanisme *fast-failover*, dapat diketahui nilai *packet loss* dari rata-rata jumlah koneksi pada masing-masing jumlah *link* yang diputus. *link* ke 1 mempunyai *packet loss* 0.71609%, *link* ke 2 mempunyai rata-rata packet loss 0.5221% dan link ke 3 mempunyai rata-rata *packet loss* 0.8839%.

Sedangkan pengujian *packet loss* pada mekanisme *fail path (recovery)*, dapat diketahui nilai *packet loss* dari rata-rata jumlah koneksi di tiap jumlah *link* yang diputus. *link* ke 1 mempunyai packet loss 0.9641%, *link* ke-2 0.873% dan *link* ke-3 1.123%.

Nilai *packet loss* pada kedua mekanisme tersebut tergolong mempunyai *packet loss* yang rendah. Akan tetapi *packet loss* yang dimekanisme *fast-failover* memunyai nilai yang rendah dibandingkan dengan mekanisme *fail path (recovery)*. Karena pada saat pemindahan jalur ke jalur yang lain mekanisme *fast failover* sudah menyediakan *backup link* tanpa komunikasi lagi dengan *controller* yang menyebabkan *packet loss* rendah, sedangkan pada mekanisme *fail path (recovery)* sistem akan berkomunikasi dengan *controller* bila ada *link* yang putus dan *controller* menemukan *link* baru tersebut yang menyebabkan *packet loss* tinggi dibandingkan meknisme *fast-failover*.