BAB 6
PENGUJIAN DAN ANALISIS

6.1 Pengujian

6.1.1 Pengujian Pencarian Jalur

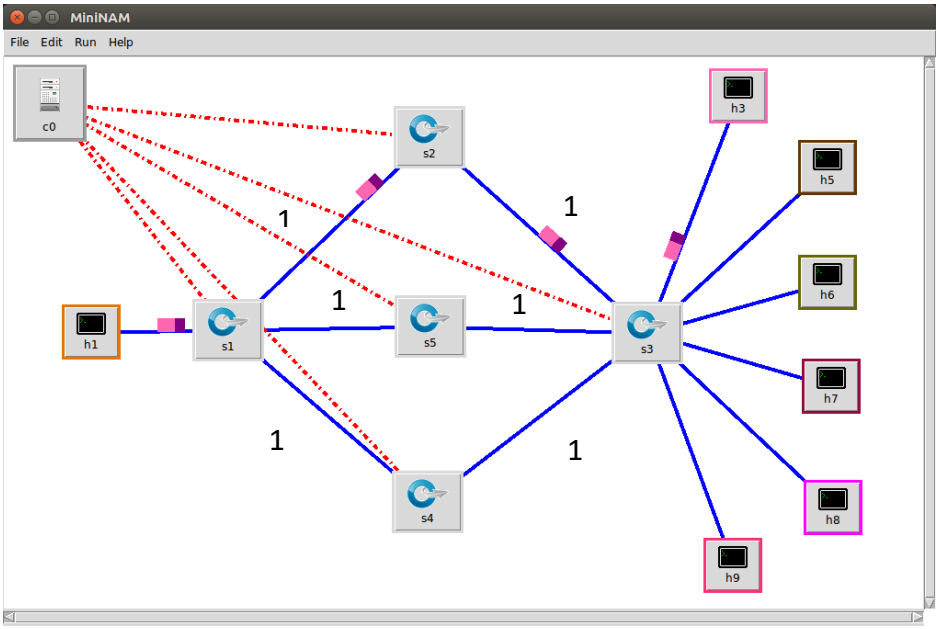
Pengujian pencarian jalur dilakukan untuk mengetahui apakah sistem telah berhasil memenuhi kebutuhan fungsionalitasnya yaitu mencari jalur terpendek pada suatu topologi dengan algoritme yang telah diimplementasikan. Pengujian ini akan dilakukan pada Topologi-1. Pada Topologi-1 terdapat 7 host, H1 akan bertindak sebagai server dan host yang lain akan bertindak sebagai *client*. Namun pada pengujian ini, hanya menggunakan 3 host saja sebagai *client* dikarenakan sudah cukup untuk mengetahui pencarian jalur pada topologi tersebut.

Untuk mengetahui jalur yang dilewati oleh *client*, *client* akan mengirimkan paket sebesar-besarnya menuju server secara bersamaan dengan jeda waktu pengiriman yang berbeda. Tools yang digunakan berupa MiniNAM, Iperf dan Bwm-ng. Pada MiniNAM, paket yang dikirimkan akan terlihat, sehingga dapat diketahui jalur yang dilewati dari paket tersebut. Kemudian Iperf digunakan untuk membuat paket sebesar-besarnya yang akan dikirimkan ke server. Sedangkan Bwm-ng digunakan untuk mengetahui *throughput* pada *outport* switch. Hasil dari pengujian pencarian jalur tersaji pada gambar dan tabel berikut ini.

Tabel 6.1 Hasil Pencarian Jalur

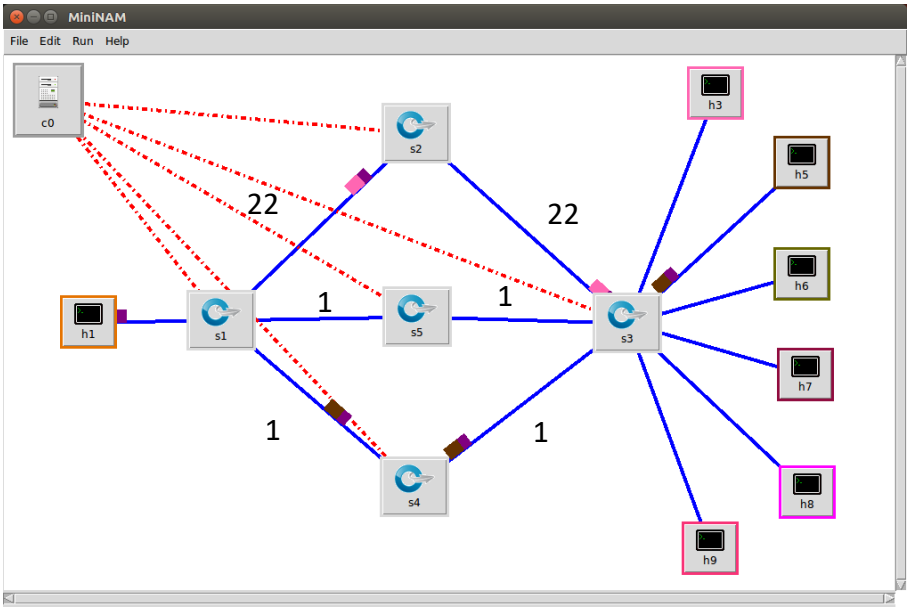
Algoritme	Jalur yg di lewati *client* ke-		
	1	2	3
Dijkstra dengan *Fuzzy* Tsukamoto	s3, s2 , s1	s3, s4, s1	s3, s5, s1
Dijkstra dengan *Fuzzy* Mamdani	s3, s2 , s1	s3, s4, s1	s3, s5, s1
Static Cost Dijkstra	s3, s2, s1	s3, s2, s1	s3, s2, s1

Pada tabel di atas merupakan hasil pencarian jalur yang dilakukan pada Topologi-1. Pengujian pencarian jalur tersebut hanya menggunakan 3 *client*. S1, s2 ,s3 dan s4 pada tabel di atas merupakan representasi dari switch pada Topologi-1 yang akan dilewati oleh *client* tersebut. Hasil dari pengujian tersebut yakni urutan switch yang dilewati oleh *client* menuju server.



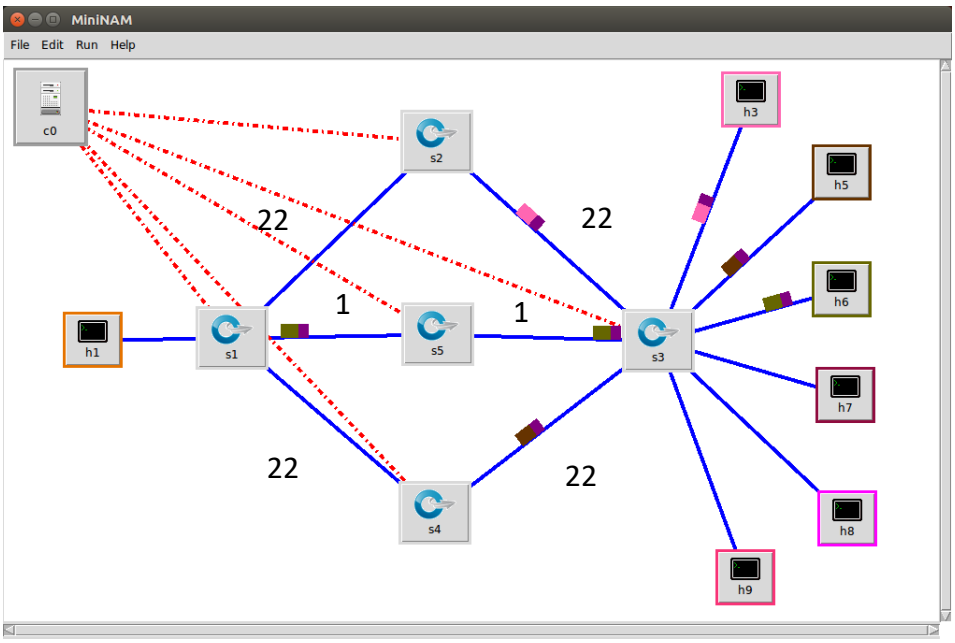
Gambar 6.1 Pengujian jalur *client* ke-1

Pada saat *client* pertama (h3) mengirimkan paket menuju server (h1), maka controller akan menghitung bobot tiap *link* berdasarkan algoritme yang telah dibuat. Pada gambar di atas merupakan contoh penentuan bobot menggunakan Logika *Fuzzy* yang akan diuji pada bagian berikutnya. Kemudian dengan menggunakan algoritme *Dijkstra*, controller akan mencari jalur terpendek yang akan dilewati paket tersebut. Dalam hal ini jalur yang dipilih adalah melalui s3 , s2, s1.



Gambar 6.2 Pengujian jalur *client* ke-2

Pada saat *client* kedua (h5) mengirimkan paket menuju server (h1), maka controller akan menghitung bobot tiap *link* berdasarkan algoritme yang telah dibuat. Pada gambar di atas merupakan contoh penentuan bobot menggunakan Logika *Fuzzy* yang akan diuji pada bagian berikutnya. Kemudian dengan menggunakan algoritme *Dijkstra*, controller akan mencari jalur terpendek yang akan dilewati paket tersebut. Dalam hal ini jalur yang dipilih adalah melalui s3 , s4, s1.



Gambar 6.3 Pengujian jalur *client* ke-3

Pada saat *client* ketiga (h6) mengirimkan paket menuju server (h1), maka controller akan menghitung bobot tiap *link* berdasarkan algoritme yang telah dibuat. Pada gambar di atas merupakan contoh penentuan bobot menggunakan Logika *Fuzzy* yang akan diuji pada bagian berikutnya. Kemudian dengan menggunakan algoritme *Dijkstra*, controller akan mencari jalur terpendek yang akan dilewati paket tersebut. Dalam hal ini jalur yang dipilih adalah melalui s3 , s5, s1. Kemudian akan dilakukan pengujian *throughput* pada outport s3 seperti pada tabel berikut.

Tabel 6.2 Hasil pengujian *throughput* pada *outport* s3

Algoritme	Rata-rata *throughput* (MB / s)			Standar Deviasi
	S3-eth5	S3-eth6	S3-eth7	
Dijkstra dengan *Fuzzy* Tsukamoto	118.24	118.23	117.43	0.38
Dijkstra dengan *Fuzzy* Mamdani	118.03	117.73	117.88	0.12
Static Cost Dijkstra	117.34	0.00	0.00	55.32

Pada tabel 5.2 merupakan hasil rata-rata *throughput* pada *outport* switch-3 (s3). Dari data tersebut dapat dilihat bahwa *throughput* dari setiap *outport* memiliki nilai yang hampir seragam yang dapat dilihat dari standar deviasi yang kecil. Hal ini berarti paket yang dikirimkan berhasil di forward menuju *outport* yang sesuai dan meratakan beban dengan algoritme yang telah dibuat.

6.1.2 Pengujian *Throughput*

Pengujian *throughput* dilakukan untuk mengetahui kualitas jaringan pada sistem dan juga untuk mengetahui apakah sistem yang telah dibuat berhasil mengatasi kelemahan pada algoritme *routing* jaringan konvensional seperti algoritme *Static Cost Dijkstra*. Pengujian *throughput* akan dilakukan pada Topologi-1 dan Topologi-2. Pada masing-masing topologi terdapat 5 host sebagai *client* dan 1 host sebagai server pada ujung topologi.

Untuk mendapatkan nilai *throughput*, *client* akan mengirimkan paket TCP sebesar-besarnya menuju server secara bersamaan dan diberi variabel pembedanya yaitu jumlah *client* dengan tujuan untuk mengetahui pengaruh jumlah *client* terhadap perubahan *throughput* yang didapatkan. Untuk melakukan pengujian ini, digunakan tools iperf dengan command “iperf -s” pada sisi server dan command “iperf -c [ip server] [parameter]” pada sisi *client*. Pada sisi *client* parameter yang digunakan yaitu durasi pengiriman sebesar 15 detik dan jumlah *client* yang terhubung ke server. Hasil dari pengujian *throughput* tersaji dalam tabel dan gambar berikut ini.

Tabel 6.3 Hasil Pengujian *Throughput* Topologi-1

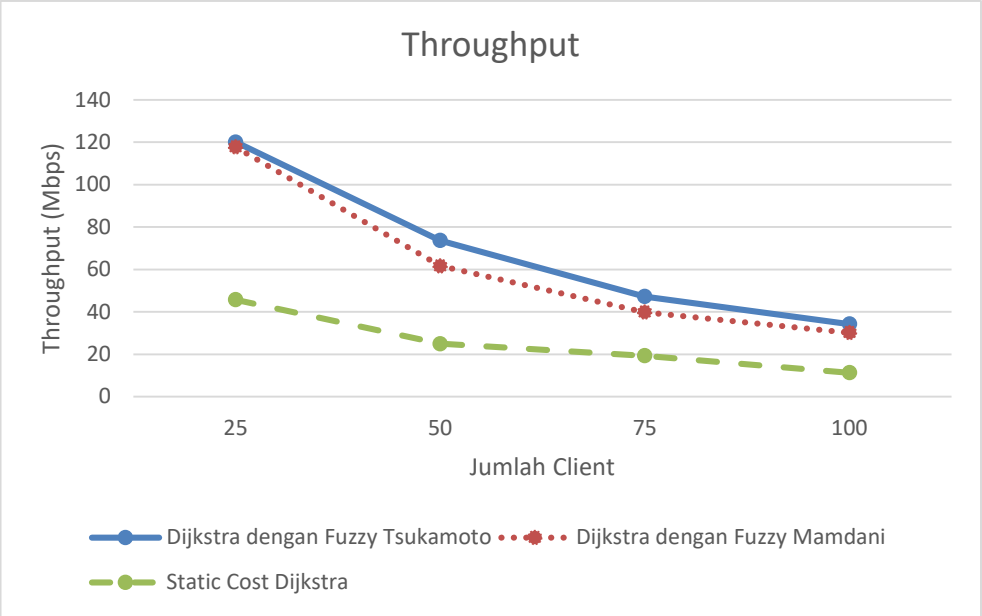
Jumlah *Client*	Rata-Rata *Throughput* (Mbit / s)		
	Dijkstra dengan *Fuzzy* Tsukamoto	Dijkstra dengan *Fuzzy* Mamdani	*Static Cost Dijkstra*
25	120.10	117.75	45.72
50	73.67	61.61	24.96
75	47.25	39.81	19.30
100	34.26	30.14	11.07

Pada tabel di atas merupakan hasil pengujian *throughput* pada Topologi-1. Masing-masing algoritme akan mendapatkan perlakuan yang sama dalam melakukan pengujian dengan variabel pembedanya yaitu jumlah *client*. Variasi dari jumlah *client* sebanyak 25, 50 ,75 dan 100. Untuk hasil pengujian *throughput* pada Topologi-2, tersaji dalam tabel berikut.

Tabel 6.4 Hasil Pengujian *Throughput* Topologi-2

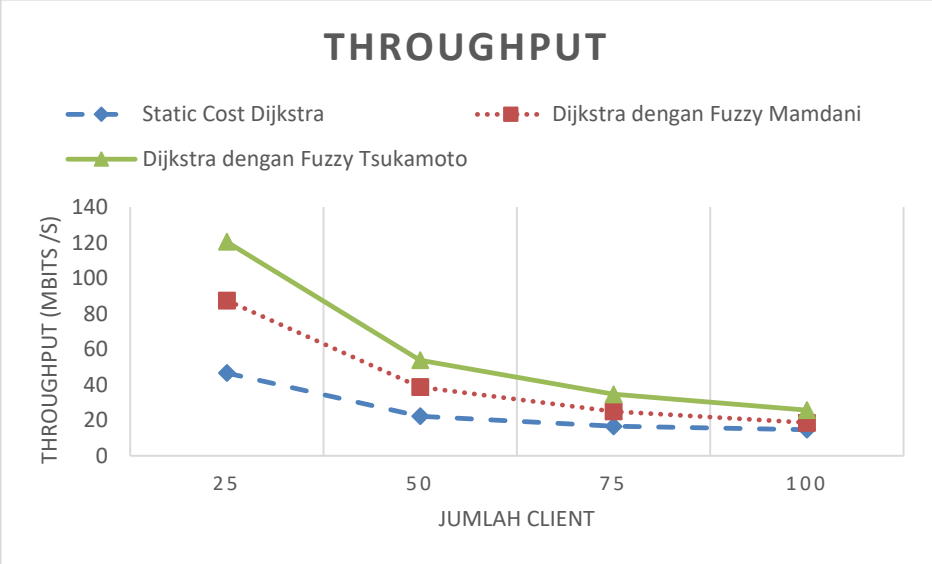
Jumlah *Client*	Rata-Rata *Throughput* (Mbit / s)		
	Dijkstra dengan *Fuzzy* Tsukamoto	Dijkstra dengan *Fuzzy* Mamdani	*Static Cost Dijkstra*
25	120.58	87.45	46.60
50	53.79	38.70	22.27
75	34.61	24.98	16.61
100	25.23	18.58	14.70

Pada tabel di atas merupakan hasil pengujian *throughput* pada Topologi-2. Masing-masing algoritme akan mendapatkan perlakuan yang sama dalam melakukan pengujian dengan variabel pembedanya yaitu jumlah *client*. Variasi dari jumlah *client* sebanyak 25, 50 ,75 dan 100. Kemudian dari tabel hasil pengujian Topologi-1 dan Topologi-2 di atas, akan disajikan dalam bentuk grafik seperti pada gambar di bawah ini.



Gambar 6.4 Grafik Perbandingan *Throughput* Topologi-1

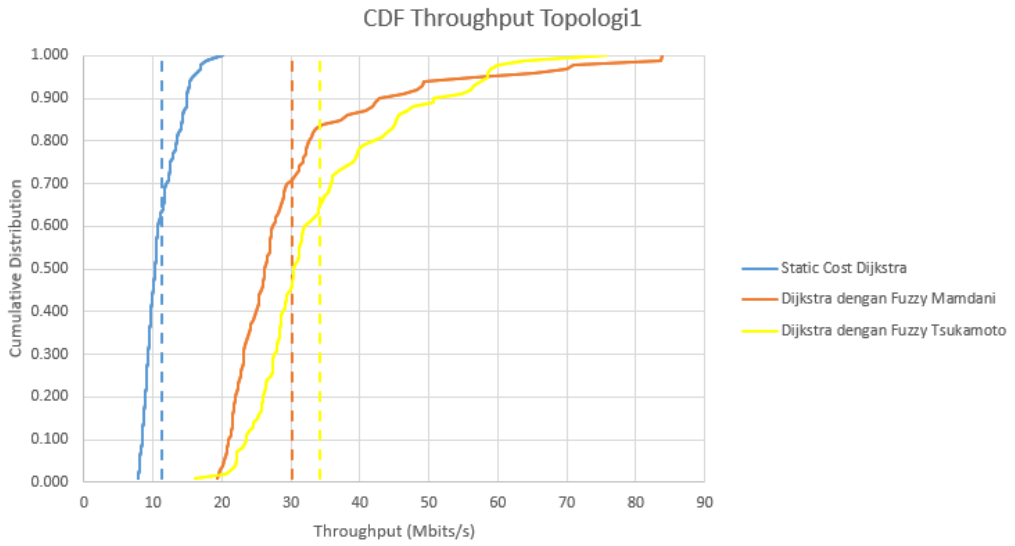
Pada gambar di atas merupakan grafik perbandingan *throughput* yang ada pada Topologi-1. Terlihat bahwa ada 3 garis yang merepresentasikan *throughput* masing-masing algoritme dengan variabel pembedanya yaitu berupa jumlah *client*. Kemudian untuk grafik perbandingan *throughput* pada Topologi-2, tersaji dalam gambar berikut ini.



Gambar 6.5 Grafik Perbandingan *Throughput* Topologi-2

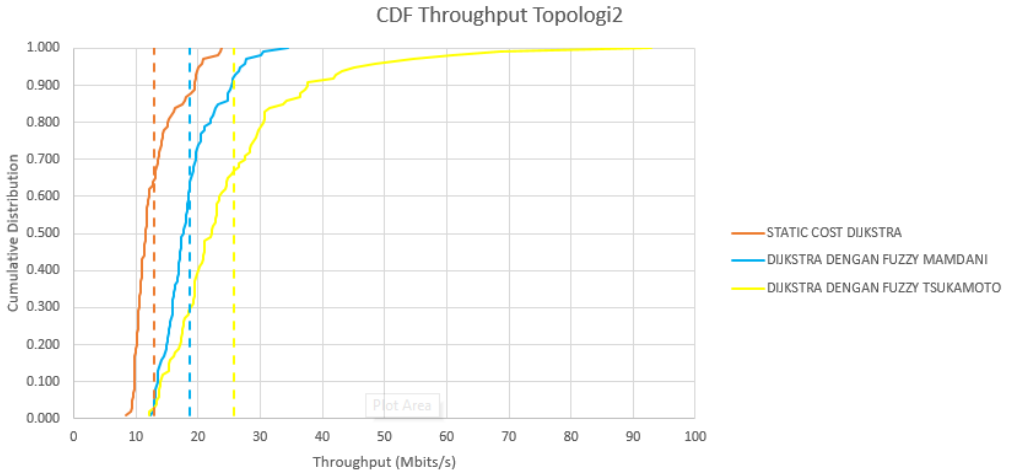
Pada gambar di atas merupakan grafik perbandingan *throughput* yang ada pada Topologi-2. Terdapat 3 garis yang merepresentasikan *throughput* masing-masing algoritme dengan variabel pembedanya yaitu berupa jumlah *client*.

Kemudian dari data *throughput* di atas, dapat dicari persebaran *throughput* untuk setiap *client* dengan menggunakan Cumulative Distribution Function (CDF). Dalam hal ini, data yang akan dicari persebarannya adalah data dengan jumlah *client* sebesar 100. Berikut ini merupakan gambar CDF *throughput* pada Topologi-1 dan Topologi-2.



Gambar 6.6 CDF Troughput Topologi-1

Pada gambar 6.6 merupakan grafik CDF yang ada pada Topologi-1. Garis putus-putus yang ada pada gambar di atas merupakan rata-rata *throughput* pada setiap algoritme yang dibedakan berdasarkan dengan warna garisnya.



Gambar 6.7 CDF Troughput Topologi-2

Pada gambar di atas merupakan grafik CDF yang ada pada Topologi-2. Garis putus-putus yang ada pada gambar di atas merupakan rata-rata *throughput* pada setiap algoritme yang dibedakan berdasarkan dengan warna garisnya. Kemudian dari gambar di atas, dapat dihitung jumlah *client* yang mendapatkan *throughput* di atas rata-rata seperti pada tabel berikut ini.

Tabel 6.5 Jumlah *Client* di atas rata-rata

Topologi ke-	Jumlah *client* di atas rata-rata		
	Static Cost Dijkstra	Dijkstra dengan *Fuzzy Mamdani*	Dijkstra dengan *Fuzzy Tsukamoto*
Topologi-1	38	29	35
Topologi-2	35	39	35

Pada tabel di atas merupakan jumlah *client* yang mendapatkan *throughput* di atas rata-rata pada 100 *client*. Jumlah *client* tersebut didapatkan dengan cara mengkalikan perpotongan garis rata-rata dengan nilai cumulative nya.

6.1.3 Pengujian *Packet Loss*

Pengujian *packet loss* dilakukan untuk mengetahui persentase kegagalan dalam mengirimkan suatu paket dari *client* ke *server* dan juga untuk mengetahui perbandingan *packet loss* pada sistem yang telah dibuat dibandingkan dengan algoritme *routing* jaringan konvensional seperti algoritme *Static Cost Dijkstra*. Pengujian *packet loss* akan dilakukan pada Topologi-1 dan Topologi-2. Pada masing-masing topologi terdapat 5 host sebagai *client* dan 1 host sebagai server pada ujung topologi.

Untuk mendapatkan nilai *packet loss*, *client* akan mengirimkan paket UDP secara bersamaan dengan transfer rate sebesar 30 Mbits/s selama 15 detik dan diberi variabel pembedanya yaitu berupa jumlah *client* dengan tujuan untuk mengetahui pengaruh jumlah *client* terhadap nilai *packet loss* yang didapatkan. Untuk melakukan pengujian ini, digunakan tools iperf dengan command “iperf -s -u” pada sisi server dan command “iperf -c [ip server] -u -P [jumlah_*client*] -b 30m -t 15” pada sisi *client*. Hasil dari pengujian *packet loss* tersaji dalam tabel dan gambar berikut ini.

Tabel 6.6 Hasil Pengujian Packet Loss Topologi-1

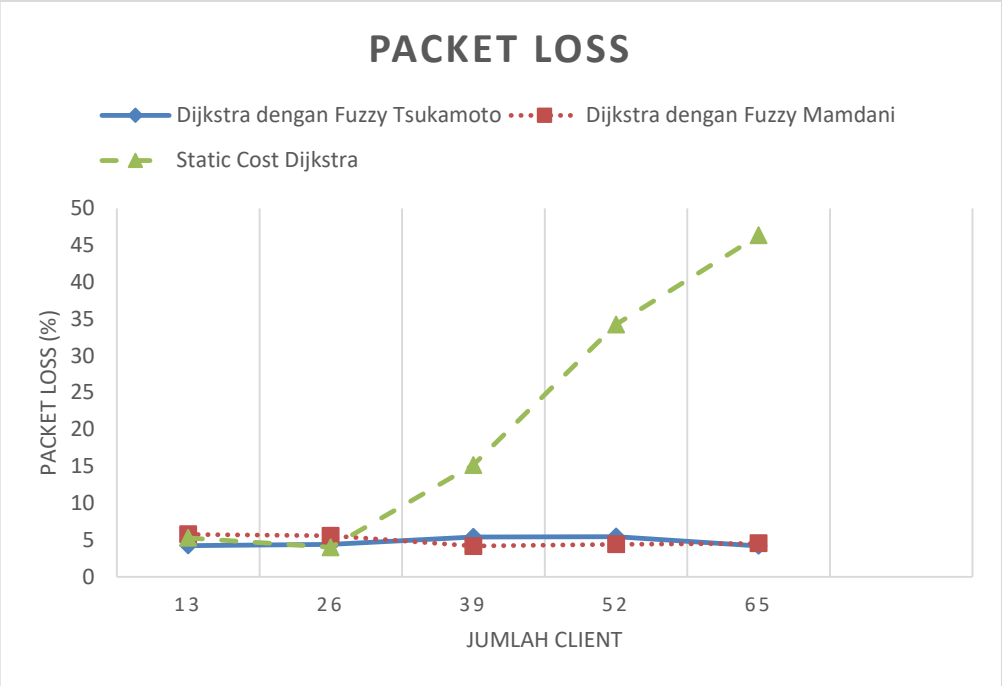
Jumlah *Client*	Rata-Rata Packet Loss (%)		
	Dijkstra dengan *Fuzzy* Tsukamoto	Dijkstra dengan *Fuzzy* Mamdani	*Static Cost Dijkstra*
13	4.24	5.76	5.28
26	4.41	5.57	4
39	5.42	4.17	15.1
52	5.46	4.43	34.2
65	4.21	4.55	46.3

Pada tabel di atas merupakan hasil pengujian *packet loss* pada Topologi-1. Masing-masing algoritme akan mendapatkan perlakuan yang sama dalam melakukan pengujian dengan variabel pembedanya yaitu jumlah *client*. Variasi dari jumlah *client*nya yaitu sebanyak 13, 26 ,39, 52 dan 65. Selanjutnya untuk hasil pengujian *packet loss* Topologi-2, tersaji dalam tabel di bawah ini.

Tabel 6.7 Hasil Pengujian Packet Loss Topologi-2

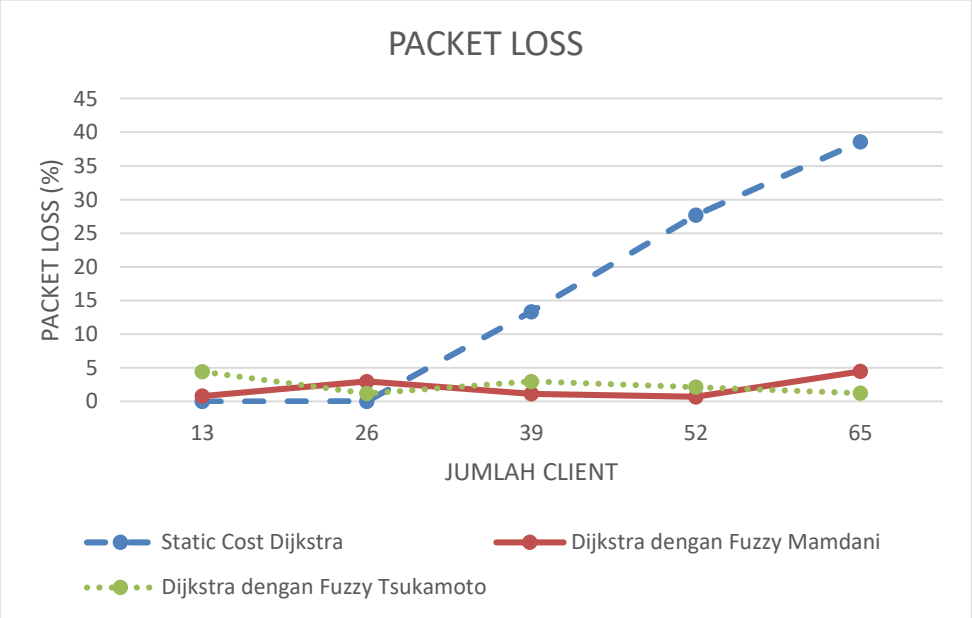
Jumlah *Client*	Rata-Rata Packet Loss (%)		
	Dijkstra dengan *Fuzzy* Tsukamoto	Dijkstra dengan *Fuzzy* Mamdani	*Static Cost Dijkstra*
13	4.43	0.81	0.01
26	1.19	2.98	0.03
39	2.96	1.14	13.35
52	2.13	0.68	27.69
65	1.23	4.47	38.61

Pada tabel di atas merupakan hasil pengujian *packet loss* pada Topologi-2. Masing-masing algoritme akan mendapatkan perlakuan yang sama dalam melakukan pengujian dengan variabel pembedanya yaitu jumlah *client*. Variasi dari jumlah *client*nya yaitu sebanyak 13, 26 ,39, 52 dan 65. Kemudian dari hasil pengujian *packet loss* pada Topologi-1 dan Topologi-2, tersaji dalam grafik pada gambar di bawah ini.



Gambar 6.8 Grafik Perbandingan Packet Loss Topologi-1

Pada gambar 6.8 merupakan grafik perbandingan *packet loss* yang ada pada Topologi-1. Terlihat bahwa ada 3 garis yang merepresentasikan *packet loss* masing-masing algoritme dengan variabel pembedanya yaitu berupa jumlah *client*. Jumlah *client* yang digunakan yakni 13, 26, 39, 52 dan 65. Kemudian untuk grafik perbandingan *packet loss* pada Topologi-2, tersaji dalam gambar di bawah ini.

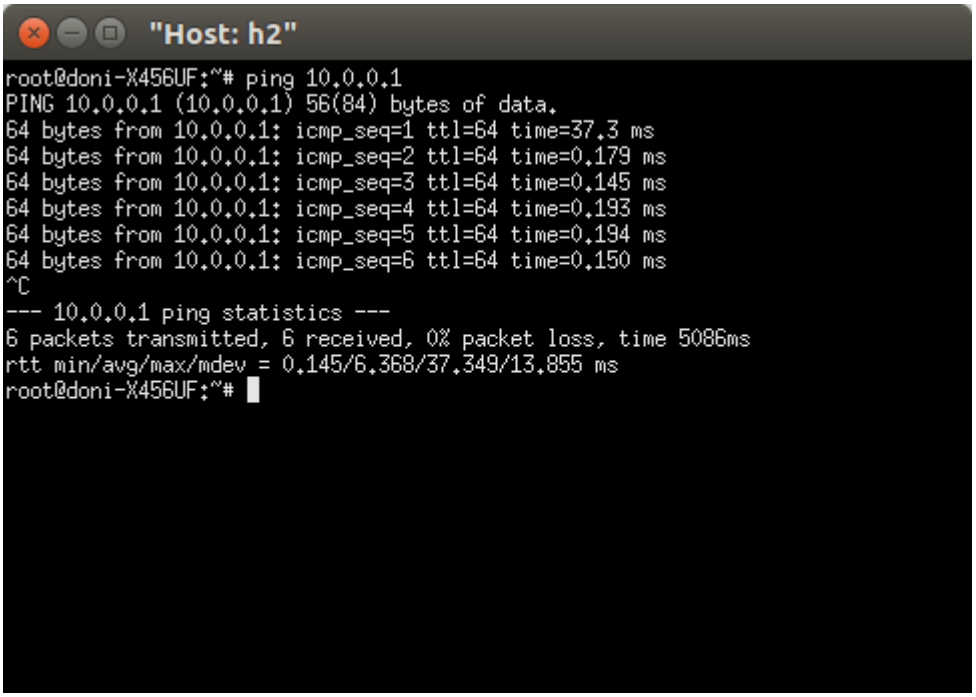


Gambar 6.9 Grafik Perbandingan Packet Loss Topologi 2

Pada gambar di atas merupakan grafik perbandingan *packet loss* yang ada pada Topologi-2. Terlihat bahwa ada 3 garis yang merepresentasikan *packet loss* masing-masing algoritme dengan variabel pembedanya yaitu berupa jumlah *client*. Jumlah *client* yang digunakan yakni 13, 26, 39, 52 dan 65.

6.1.4 Pengujian Convergence Time

Pengujian Convergence Time dilakukan untuk mengetahui waktu pemrosesan dari suatu algoritme *routing*. Pengujian *convergence time* akan dilakukan pada Topologi-1 dan Topologi-2. Untuk mendapatkan nilai *convergence time*, dapat dilakukan dengan cara mengirimkan paket ICMP dari *client* menuju server yang berada pada ujung topologi. Kemudian timer pada program controller akan berjalan menghitung waktu pemrosesan algoritme tersebut. Pengujian dilakukan sebanyak 10 kali untuk setiap algoritme. Hasil dari pengujian *convergence time* tersaji pada tabel berikut ini.



Gambar 6.10 *Client* mengirim paket ICMP menuju server

Pada gambar di atas merupakan screenshot dari *client* yang mengirimkan paket ICMP menuju server untuk memicu controller menjalankan algoritme pencarian jalur. Kemudian controller akan memulai timer yang digunakan sebagai perhitungan *convergence time*.

Tabel 6.8 Hasil Pengujian Convergence Time

Topologi ke-	Convergence Time (ms)					
	Dijkstra dengan *Fuzzy* Tsukamoto		Dijkstra dengan *Fuzzy* Mamdani		*Static Cost Dijkstra*	
	Rata-rata	Standar Deviasi	Rata-rata	Standar Deviasi	Rata-rata	Standar Deviasi
1	1.08	0.78	20.14	10.64	0.10	0.05
2	2.47	1.58	44.07	7.68	0.20	0.15

Pada tabel di atas merupakan hasil pengujian *convergence time*. Convergence time akan diujikan pada Topologi-1 dan Topologi-2. Dari hasil pengujian, akan dihitung nilai rata-rata dan standar deviasi sebagai acuan dalam penarikan kesimpulan.

6.1.5 Pengujian Logika *Fuzzy*

Pengujian logika *fuzzy* dilakukan untuk mengetahui ketepatan dari penentuan bobot *link* menggunakan perhitungan *fuzzy* dari sistem yang telah dibuat dibandingkan dengan hasil perhitungan manual yang merujuk pada dasar teori pada bagian sebelumnya. Pengujian ini dilakukan hanya pada Topologi-1.

Untuk melakukan pengujian ini, *client* akan mengirim paket UDP menuju server yang bertetangga dengan *client* tersebut dan akan diberikan variabel pembedanya yaitu beban *traffic* dengan tujuan untuk mengetahui pengaruh beban *traffic* terhadap hasil pengujian ini. Digunakan tools Iperf untuk mengirimkan paket dari *client* menuju server. Controller akan menampilkan hasil perhitungan bobot *link* berdasarkan dengan perhitungan logika *fuzzy* yang telah dibuat. Kemudian nilai error didapatkan seperti pada persamaan 6.1 berikut.

$$\text{Error} = \frac{|Output\ Sistem - Output\ Manual\ |}{Output\ Manual} x\ 100 \qquad (6.1)$$

Nilai error didapatkan dengan cara menghitung nilai selisih antara output sistem dan output manual kemudian dibagi dengan nilai output manual. Hasil dari pengujian logika fuzzy tersaji pada tabel di bawah ini.

Tabel 6.9 Hasil Pengujian Logika *Fuzzy* Tsukamoto

No	*Beban* (Mbits/s)	*Traffic* terpakai (%)	*Delay* (ms)	*Output* Sistem	*Output* Manual	Error (%)
1	100	10	3	0	0	0
2	200	20	7	0	0	0
3	300	30	5	11.25	11.25	0
4	400	41	1	15	15	0
5	500	52	1	15	15	0
6	600	63	2	14.21	14.21	0
7	700	75	1	17.81	17.81	0
8	800	83	2	22.5	22.5	0
9	900	92	1	22.5	22.5	0
10	1000	98	13	22.5	22.5	0
Rata-Rata Error						0

Pada tabel 6.8 merupakan hasil pengujian pada logika *fuzzy Tsukamoto*. Pengujian dilakukan sebanyak 10 kali dengan variasi beban traffic yang berbeda-beda. Kemudian sistem akan mendeteksi *traffic* dan *delay* yang digunakan sebagai input pada logika *fuzzy*. Selanjutnya sistem akan menampilkan bobot sesuai dengan perancangan yang telah dibuat sebelumnya.

Tabel 6.10 Hasil Pengujian Logika *Fuzzy* Mamdani

No	*Traffic* (Mbits/s)	*Traffic* terpakai (%)	*Delay* (ms)	*Output* Sistem	*Output* Manual	Error (%)
1	100	10	1	5,83	4,5	29.5
2	200	20	5	5,83	4,5	29.5
3	300	30	7	10,35	10	3.5
4	400	41	1	14.99	15	0.06
5	500	52	1	14.99	15	0.06
6	600	63	1	16,54	17,09	3.2
7	700	75	1	21,81	21,81	0
8	800	83	1	24,16	26	7.07
9	900	92	2	24,16	26	7.07
10	1000	98	13	24,16	26	7.07
Rata-Rata Error						8.7

Pada tabel di atas merupakan hasil pengujian pada logika *fuzzy Mamdani*. Pengujian dilakukan sebanyak 10 kali dengan variasi beban yang berbeda-beda. Kemudian sistem akan mendeteksi *traffic* dan *delay* yang digunakan sebagai input pada logika *fuzzy*. Selanjutnya sistem akan menampilkan bobot sesuai dengan perancangan yang telah dibuat sebelumnya.

6.2 Analisis

6.2.1 Analisis Pencarian Jalur

Berdasakan hasil pengujian pencarian jalur yang telah dilakukan, sistem telah berhasil mencari jalur terpendek berdasarkan algoritme pencarian jalur yang telah dibuat. Pada algoritme *Dijkstra* dengan *fuzzy* Tsukamoto, ke 3 *client* melewati jalur yang berbeda untuk setiap *client*nya, yakni *client* pertama melewati jalur (s3, s2, s1), *client* kedua melewati jalur (s3,s4,s1) dan *client* ketiga melewati jalur (s3,s5,s1). Pada algoritme *Dijkstra* dengan *fuzzy* Mamdani, ke 3 *client* melewati jalur yang sama dengan algoritme *Dijkstra* dengan *fuzzy* Tsukamoto, yakni (s3, s2, s1), (s3, s4, s1), dan (s3, s5, s1). Sedangkan pada algoritme *Static Cost Dijkstra*, ke 3 *client* melewati jalur yang sama untuk setiap *client* nya yaitu s3,s2,s1. Hal ini dikarenakan pada algoritme *Static Cost Dijkstra* semua *link* memiliki bobot yang sama dan bernilai tetap sehingga *controller* hanya memilih salah satu jalur saja.

Kemudian dari hasil pengujian *throughput* pada *outport* s3, pada algoritme *Dijkstra* dengan *fuzzy* Tsukamoto memiliki nilai *throughput* yang tidak berbeda secara signifikan untuk setiap *outport* nya dengan standart deviasi sebesar 0.38 . Kemudian pada algoritme *Dijkstra* dengan *fuzzy* Mamdani juga memiliki nilai *throughput* yang tidak berbeda secara signifikan untuk setiap *outport* nya dengan standart deviasi sebesar 0.1 . Sedangkan pada algoritme *Static Cost Dijkstra*, hanya satu *outport* yang memiliki nilai *throughput* sedangkan *outport* yang lain bernilai 0 dengan standar deviasi ketiga *outport* sebesar 55. Hal ini berarti pada algoritme *Dijkstra* dengan *fuzzy* Tsukamoto dan Mamdani, controller telah berhasil mencari jalur terpendek berdasarkan algoritme yang telah dibuat dan meratakan beban ke semua *outport*. Sedangkan pada algoritme *Static Cost Dijkstra* , controller juga telah berhasil mencari jalur terpendek namun tidak meratakan beban ke semua *outport* yakni hanya menggunakan satu *outport* saja.

6.2.2 Analisis *Throughput*

Berdasarkan hasil pengujian *throughput* yang telah dilakukan, nilai *throughput* akan berkurang seiring dengan bertambahnya jumlah *client* yang terhubung ke server. Pada Algoritme *Static Cost Dijkstra* memiliki *throughput* paling buruk dibandingkan dengan algoritme yang lain baik pada Topologi-1 ataupun Topologi-2. Hal ini dikarenakan pada algoritme *Static Cost Dijkstra* memiliki jalur yang tetap untuk setiap *client*nya. Sedangkan pada algoritme

Dijkstra dengan *fuzzy* Tsukamoto dan algoritme *Dijkstra* dengan Mamdani memiliki *throughput* yang lebih unggul dibandingkan dengan algoritme *Static Cost Dijkstra*, baik pada Topologi-1 ataupun Topologi-2. Namun pada algoritme *Dijkstra* dengan *fuzzy* Tsukamoto memiliki *throughput* paling bagus di antara algoritme yang lain. Hal ini dikarenakan pada algoritme dijsktra dengan *fuzzy* Tsukamoto menggunakan jalur yang berbeda-beda untuk setiap *client*nya dan memiliki *convergence time* lebih baik dibandingkan dengan algoritme *Dijkstra* dengan *fuzzy* Mamdani.

Kemudian berdasarkan dari pengujian CDF Topologi-1 dan Topologi-2, dapat dilihat persebaran *throughput* yang didapatkan untuk setiap *client*. Grafik CDF yang ideal adalah yang berbentuk mendekati garis lurus yang berarti pendistribusiannya merata. Pada Topologi-1, algoritme *Static Cost Dijkstra* memiliki grafik CDF yang ideal dikarenakan grafiknya mendekati garis lurus. Sedangkan pada algoritme *Dijkstra* dengan *fuzzy* Mamdani dan Tsukamoto memiliki persebaran yang tidak merata dibuktikan dengan grafiknya yang jauh dari garis lurus. Kemudian pada Topologi-2, algoritme *Static Cost Dijkstra* dan algoritme *Dijkstra* dengan *fuzzy* Mamdani memiliki persebaran yang cukup merata dibuktikan dengan bentuk grafik yang mendekati garis lurus. Sedangkan pada algoritme *Dijkstra* dengan *fuzzy* Tsukamoto memiliki persebaran yang tidak merata dengan bentuk grafik menjauh dari garis lurus. Hal ini bisa terjadi karena pada algoritme *Dijkstra* dengan *fuzzy* Tsukamoto dan Mamdani memiliki jalur yang berbeda-beda untuk setiap *client* nya menyebabkan porsi *throughput* yang didapat oleh client tidak merata.

6.2.3 Analisis Packet Loss

Berdasarkan hasil pengujian *packet loss*, pada algoritme *Static Cost Dijkstra* akan mengalami kenaikan *packet loss* yang signifikan dibandingkan dengan algoritme lain pada saat jumlah *client* lebih dari 39 baik pada Topologi-1 ataupun Topologi-2. Sedangkan pada algoritme dijsktra dengan *fuzzy* Tsukamoto dan algoritme *Dijkstra* dengan *fuzzy* Mamdani memiliki nilai *packet loss* berkisar antara 0 - 5 persen pada semua kategori jumlah *client*, baik pada Topologi-1 ataupun Topologi-2. Hal ini dikarenakan pada algoritme *Static Cost Dijkstra* hanya menggunakan satu jalur saja untuk setiap *client*nya. Sedangkan pada algoritme *Dijkstra* dengan *fuzzy* Tsukamoto dan Mamdani menggunakan jalur yang berbeda-beda untuk setiap *client*nya.

6.2.4 Analisis Convergence Time

Berdasarkan hasil pengujian *convergence time* yang telah dilakukan dengan cara mengirimkan paket ICMP yang dilakukan sebanyak 10 kali pada setiap topologi, sistem telah berhasil melakukan pencarian jalur terpendek dari setiap topologi baik itu Algoritme *Static Cost Dijkstra*, Algoritme Dijkstra dengan *Fuzzy* Tsukamoto ataupun Algoritme Dijkstra dengan *Fuzzy* Mamdani. Perhitungan *convergence time* didapatkan berdasarkan waktu yang dibutuhkan oleh *routing engine* dalam pembentukan seluruh rute yang berada pada *data plane*.

Dari hasil yang di dapat untuk setiap variasi jumlah *switch* pada setiap topologi akan mengalami perbedaan *convergence time,* karena apabila dilihat dari hasil pengujian nya, *convergence time* bertambah seiring dengan penambahan *switch* pada topologi. Jika diurutkan berdasarkan jumlah *switch* pada topologi, maka Topologi-1 < Topologi-2, sama seperti perbandingan *convergence time* rata-rata dari topologi-topologi tersebut. Dari hasil pengujian juga diketahui bahwa pada semua topologi yang sudah diujikan, Algoritme *Static Cost Dijkstra* memiliki *convergence time* lebih unggul dibandingkan dengan Algoritme Dijkstra dengan *Fuzzy* Tsukamoto ataupun Algoritme *Dijkstra* dengan *Fuzzy* Mamdani. Sedangkan pada algoritme *Dijkstra* dengan *Fuzzy* Mamdani memiliki *convergence time* paling buruk dengan perbedaan yang signifikan dibandingkan dengan algoritme yang lain. Hal ini dikarenakan program controller pada algoritme *Dijkstra* dengan *Fuzzy* Mamdani menggunakan variabel dasar berupa *array* sehingga memerlukan waktu pemrosesan yang lebih lama dibandingkan dengan variabel biasa.

6.2.5 Analisis Logika *Fuzzy*

Berdasarkan hasil pengujian logika *fuzzy* yang telah dilakukan, pada *fuzzy* Tsukamoto memiliki *output* sistem yang tidak berbeda dibandingkan dengan *output* yang dihasilkan dengan perhitungan manual. Sedangkan pada *fuzzy* Mamdani memiliki *output* sistem yang memiliki perbedaan yang tidak signifikan dibandingkan dengan *output* perhitungan manual dengan tingkat error sebesar 8.7 %. Hal ini dikarenakan adanya perbedaan perhitungan pada metode centroid yang digunakan. Namun perbedaan tersebut tidak berpengaruh pada sistem dikarenakan pencarian jalur terpendek dilakukan dengan membandingkan semua bobot *link* yang ada pada topologi.