BAB 5
PERANCANGAN DAN IMPLEMENTASI

Pada bagian ini akan dibahas mengenai perancangan dan implementasi dari sistem yang akan dibuat dengan mengacu pada metodologi yang telah dibahas.

5.1 Perancangan Sistem

Pada bagian ini berisi tentang bagaimana membuat algoritme *routing* yang dapat menentukan bobot *link* secara dinamis berdasarkan *delay* dan *traffic* pada jaringan. Algoritme Dijkstra digunakan untuk melakukan *routing* dari pengirim ke penerima. Untuk menentukan bobot *link*, digunakan logika *fuzzy* dengan parameter yang digunakan yaitu *delay* dan *traffic*. Metode logika *fuzzy* yang digunakan pada sistem ini yaitu metode *Mamdani* dan *Tsukamoto*. Logika *fuzzy* dapat memetakan *input* berupa *traffic* dan *delay* menjadi satu keluaran bobot *link* yang akan digunakan dalam pemilihan jalur terpendek pada algoritme Dijkstra. Tahap perancangan dilakukan untuk melakukan perencanaan terhadap sistem yang akan dibuat yang terbagi menjadi beberapa bagian yaitu sebagai berikut.

5.1.1 Analisis Kebutuhan Perangkat Keras dan Lunak

Dalam Penelitian ini membutuhkan perangkat keras dan lunak yang digunakan untuk perancangan maupun pengujian sistem. Kebutuhan perangkat keras dan perangkat lunak yang dibutuhkan adalah sebagai berikut.

1) Kebutuhan perangkat keras yaitu 1 buah PC dengan spesifikasi:

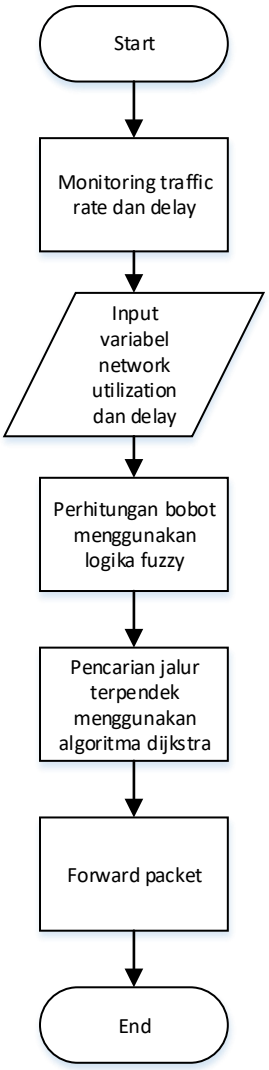
- CPU = Intel Core i5-550, 3.20GHz
- RAM = 4 GB
- Harddisk = 500 GB

2) Kebutuhan perangkat lunak, yaitu :

- Linux Ubuntu 16.04
- Mininet versi 2.2.1
- Ryu Controller
- Sflow-RT
- Library python *scikit-fuzzy*

5.1.2 Diagram Alir Sistem

Dalam perancangan sistem ini terdapat beberapa tahapan yang harus dilakukan. Berikut ini merupakan gambar diagram alir dari perancangan sistem yang dibuat.

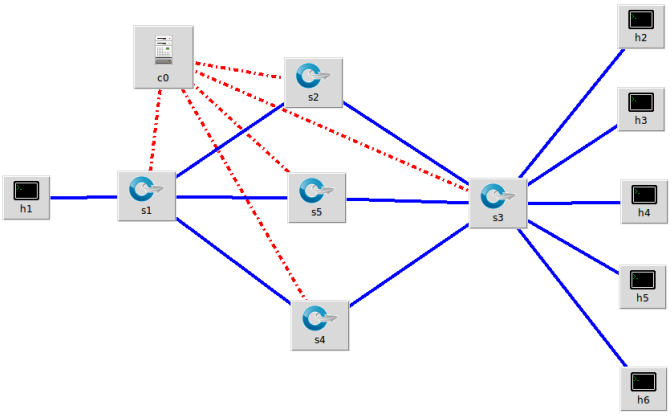


Gambar 5.1 Flowchart Keseluruhan Sistem

Pertama sistem akan melakukan proses monitoring *traffic* dan *delay* pada suatu jaringan. Kemudian variabel *traffic* dan *delay* digunakan sebagai *input* untuk menentukan bobot *link*. Logika *fuzzy* digunakan untuk menentukan bobot *link* dengan *input* berupa *traffic* dan *delay*. Kemudian setelah didapatkan bobot pada setiap *link*, maka akan dilakukan *routing* dengan mengunakan algoritme Dijkstra. Kemudian *switch* akan meneruskan paket menuju *outport* yang sesuai.

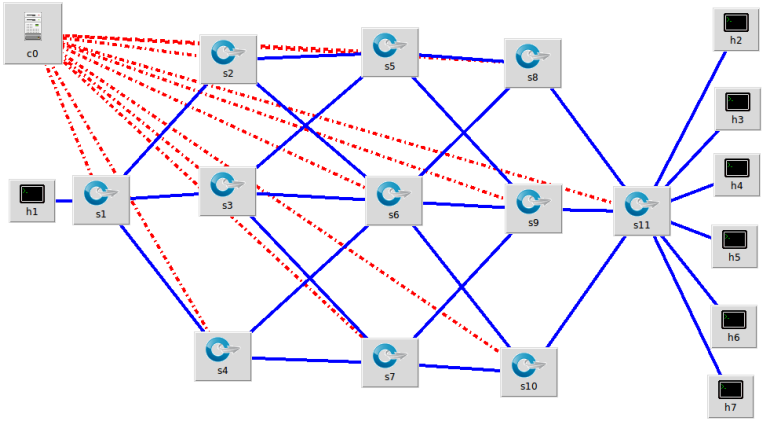
5.1.3 Topologi

Topologi digunakan untuk mengaplikasikan sistem yang telah dibuat. Topologi yang digunakan adalah topologi yang memiliki jalur yang redundan, yaitu topologi yang berbentuk *loop* dan memiliki lebih dari satu jalur dari *node* awal ke tujuan (Liveline, 2018). Terdapat 2 topologi yang digunakan pada penelitian ini yaitu Topologi-1 dan Topologi-2 seperti pada gambar di bawah ini.



Gambar 5.2 Topologi-1

Pada Topologi-1 terdapat 5 *openvswitch* dengan *bandwidth* antar *link* sebesar 1000 Mbits. Pada topologi tersebut hanya menggunakan 5 *switch* dikarenakan topologi tersebut digunakan untuk mengetahui fungsionalitas dari suatu sistem.



Gambar 5.3 Topologi-2

Pada Topologi-2 terdiri dari 11 *openvswitch* dengan *bandwidth* setiap *link* sebesar 1000 Mbits. Pada topologi di atas tergolong cukup kompleks dikarenakan memiliki banyak jalur yang digunakan untuk mengirimkan paket. Sehingga

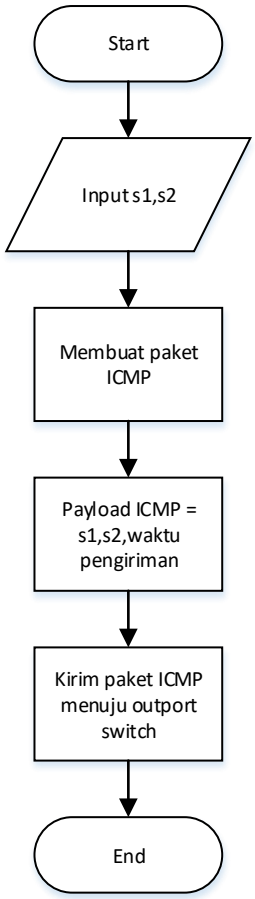
diharapkan topologi tersebut mampu mewakili topologi-topologi lain yang ada pada pada suatu jaringan.

5.1.4 Perancangan Monitoring *Traffic*

Pada bagian ini adalah bagaimana membangun sebuah fungsi yang digunakan untuk memantau *traffic* pada setiap *link*. Dengan menggunakan aplikasi sFlow-RT, *traffic* pada setiap *link* dapat diakuisisi secara real-time. *Controller* dapat mengambil nilai *traffic* tersebut dengan melakukan HTTP request GET menuju alamat *collector* sFlow-RT. Kemudian pada *controller,* nilai *traffic* akan dibandingkan dengan *bandwidth* yang ada pada *link* tersebut sehingga menghasilkan *network utilization*.

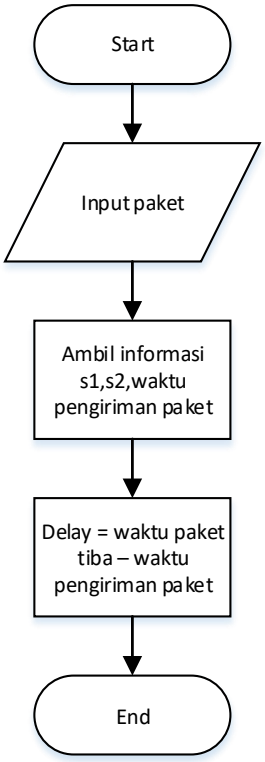
5.1.5 Perancangan Monitoring *Delay*

Pada bagian ini berisi tentang bagaimana membangun sebuah fungsi yang digunakan untuk memantau *delay* pada setiap *link.* Nilai *delay* didapatkan dengan cara mengirim paket ICMP menggunakan packet library pada Ryu. Diagram alir perancangan monitoring *delay* seperti pada gambar berikut ini.



Gambar 5.4 Membuat paket ICMP

Untuk dapat mengirimkan paket ICMP, diperlukan informasi mengenai switch yang bertetangga (“s1” dan “s2”). Kemudian controller akan membuat paket ICMP menggunakan packet library yang ada pada Ryu. *Payload* pada paket tersebut akan diisi data-data yang dibutuhkan dalam pencarian delay seperti informasi pasangan switch yang bertetangga dan waktu pengiriman paket tersebut. Kemudian paket tersebut akan dikirimkan menuju outport pada switch tersebut. Disisi lain terdapat fungsi yang digunakan untuk meng-handle paket ICMP ketika paket tersebut sampai pada switch tujuan. Diagram alir fungsi ICMP *handler* seperti pada gambar di bawah ini.

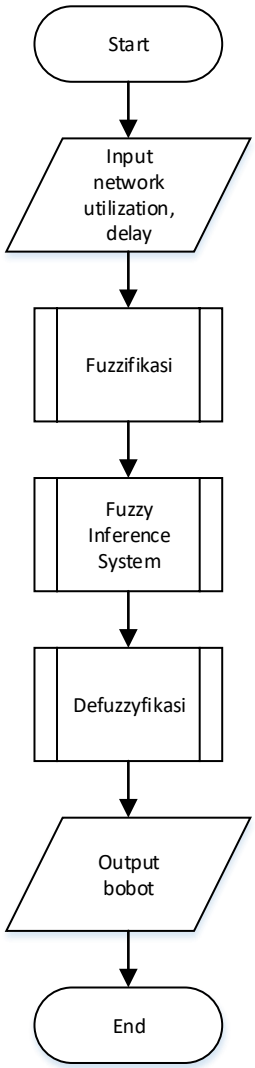


Gambar 5.5 Paket ICMP *handler*

Ketika paket ICMP masuk pada switch tujuan, maka *payload* paket tersebut akan diekstrak sehingga menghasilkan data-data yang diperlukan dalam pencarian delay seperti informasi switch yang bertetangga dan waktu pengiriman paket. Berdasarkan data tersebut, *delay* dari switch yang bertetangga (“s1” dan “s2”) didapatkan dengan cara menghitung selisih antara waktu saat paket tiba pada switch tujuan dengan waktu saat pengiriman dari switch asal.

5.1.6 Perancangan Logika *Fuzzy*

Logika *fuzzy* pada sistem ini digunakan untuk menentukan bobot pada tiap *link*. Penentuan bobot *link* tersebut yaitu berdasarkan pada *traffic* dan *delay* pada jaringan. Kedua parameter ini digunakan sebagai *input* dalam perhitungan logika *fuzzy* yang digunakan. Alur perancangan logika *fuzzy* tersaji dalam gambar berikut ini.



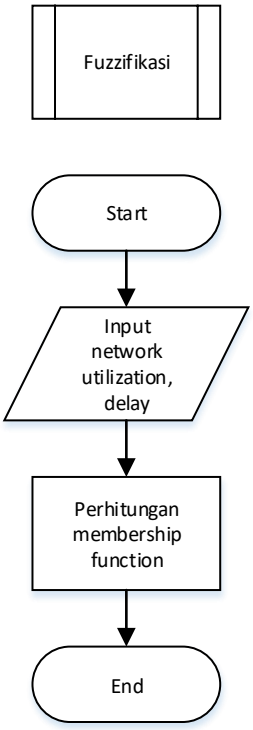
Gambar 5.6 Flowchart perancangan logika *fuzzy*

Proses perancangan logika *fuzzy* diawali dengan melakukan pembacaan nilai *traffic* dan *delay*. Selanjutnya akan memasuki tahap fuzzifikasi dengan *input* berupa *traffic* dan *delay*. Kemudian akan dilakukan proses inferensi dengan menggunakan *fuzzy inference system Mamdani* dan *fuzzy inference system Tsukamoto*. Setelah itu akan dilakukan de*fuzzy*fikasi untuk merubah dari

himpunan *fuzzy* ke dalam bentuk *crisp* yang berupa bobot. Untuk lebih jelasnya akan tersaji pada sub bab berikut ini.

5.1.6.1 Fuzzifikasi

Fuzzifikasi merupakan tahapan awal dalam proses logika *fuzzy* dimana proses ini merubah nilai yang berupa crisp menjadi ke dalam bentuk *fuzzy* yang berupa variabel linguistic dengan fungsi keanggotaan tertentu. Diagram alur fuzzifikasi tersaji pada gambar berikut ini.



Gambar 5.7 Flowchart Fuzzifikasi

Pada proses ini terdapat 3 variabel yang akan diubah menjadi bentuk bilangan *fuzzy* yaitu 2 variabel sebagai *input* dan 1 variabel sebagai *output*. Variabel *input* berupa *traffic* dan *delay*. Sedangkan variabel *output* berupa bobot yang telah ditentukan nilai-nilainya.

Pada variabel *traffic* , terdapat 3 himpunan *fuzzy* yang akan dimodelkan yakni kecil, sedang dan besar. Berikut ini merupakan tabel hubungan linguistik dan numerik pada variabel *traffic*.

Tabel 5.1 Linguistik variabel *traffic*

Linguistik	**Numerik**
Besar	Lebih dari 60 %
Sedang	20 – 80 %
Kecil	0 – 40 %

Pada tabel diatas, terdapat 3 himpunan *fuzzy* yang digunakan yaitu kecil, sedang dan besar. Hal ini mengacu pada penelitian yang telah dilakukan sebelumnya (Cisco, 2013) dimana untuk mendapatkan performa yang optimal pada suatu *link*, maka nilai *traffic* harus tidak lebih dari 80 persen. Kemudian untuk mendapatkan nilai derajat keanggotaan dari variabel *traffic*, akan tersaji pada persamaan di bawah ini.

$$\mu[x]\ \text{Kecil} = \begin{cases} 1; & x \le 20 \\ \frac{40-x}{40-20}; & 20 \le x \le 40 \\ 0; & x \ge 40 \end{cases} \quad (5.1)$$

$$\mu[x]\ \text{Sedang} = \begin{cases} 0; & x \le 20 \text{ atau } x \ge 80 \\ \frac{x-20}{40-20}; & 20 \le x \le 40 \\ 1; & 40 \le x \le 60 \\ \frac{80-X}{80-60}; & 60 \le x \le 80 \end{cases} \quad (5.2)$$

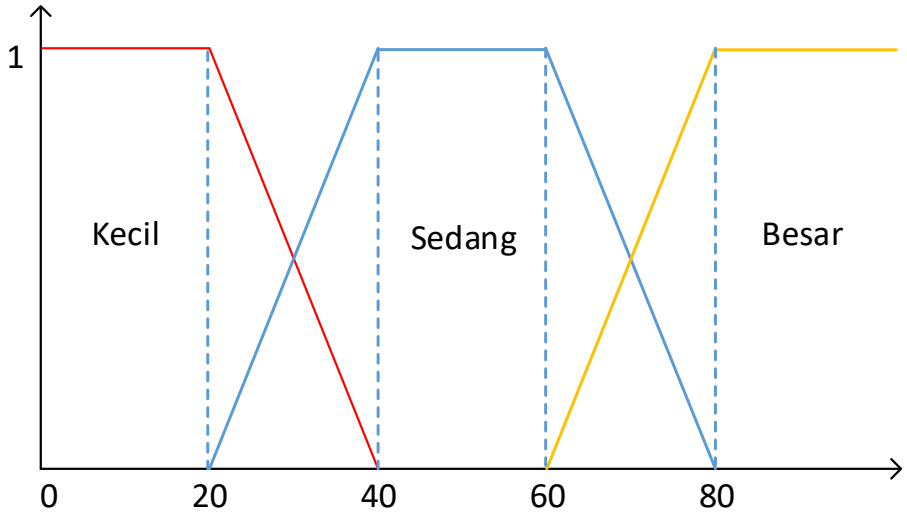
$$\mu[x]\ \text{Besar} = \begin{cases} 0; & x \le 60 \\ \frac{x-60}{80-60}; & 60 \le x \le 80 \\ 1; & x \ge 80 \end{cases} \quad (5.3)$$

Persamaan (5.1) digunakan utuk menghitung nilai derajat keanggotaan himpunan *fuzzy* kategori rendah, jika *traffic* kurang dari 20, maka derajat keanggotaannya akan bernilai 1 , kemudian jika *traffic* lebih dari 20 dan kurang dari 40 maka nilai derajat keanggotaannya akan bernilai sesuai dengan hasil persamaan diatas, apabila *traffic* lebih dari 40 dan kurang dari 60, maka derajat keanggotaannya akan bernilai 0.

Persamaan (5.2) digunakan utuk menghitung nilai derajat keanggotaan himpunan *fuzzy* kategori sedang, jika *traffic* kurang dari 20 atau lebih dari 80, maka derajat keanggotaannya akan bernilai 0 , kemudian jika *traffic* lebih dari 20 dan kurang dari 40, maka nilai derajat keanggotaannya akan bernilai sesuai dengan hasil persamaan diatas, apabila *traffic* lebih dari 40, maka derajat keanggotaannya akan bernilai 1, terakhir jika *traffic* lebih dari 60 dan kurang dari 80, maka derajat keanggotaannya akan bernilai sesuai dengan hasil persamaan diatas.

Persamaan (5.3) digunakan utuk menghitung nilai derajat keanggotaan himpunan *fuzzy* kategori besar, jika *traffic* kurang dari 60, maka derajat keanggotaannya akan bernilai 0 , kemudian jika *traffic* lebih dari 60 dan kurang dari 80 maka nilai derajat keanggotaannya akan bernilai sesuai dengan hasil persamaan diatas, apabila *traffic* lebih dari 80, maka derajat keanggotaannya akan bernilai 1.

Himpunan *fuzzy* variabel *traffic* dengan kategori kecil, sedang dan besar dapat direpresentasikan dalam sebuah grafik seperti pada gambar di bawah ini.



Gambar 5.8 Grafik derajat keanggotaan variabel *traffic*

Pada gambar di atas terdapat 3 fungsi keanggotaan. Pertama adalah kategori kecil dengan bentuk grafik membership berupa trapesium bahu kiri. Kemudian yang kedua adalah kategori sedang dengan bentuk grafik membership berupa trapesium. Terakhir adalah kategori besar dengan bentuk grafik membership berupa trapesium bahu kanan.

Kemudian pada variabel *delay* , terdapat 2 himpunan *fuzzy* yang akan dimodelkan yakni kecil dan besar. Berikut ini merupakan tabel hubungan linguistik dan numerik pada variabel *delay*.

Tabel 5.2 Fuzzifikasi variabel *delay*

Linguistik	**Numerik**
Besar	Lebih dari 300 ms
Kecil	0 - 400 ms

Pada tabel di atas, terdapat 2 kategori yang digunakan dalam variabel *delay* yaitu besar dan kecil. Data di atas mengacu pada penelitian sebelumnya (Cisco, 2006) dimana jika *delay* lebih besar dari 400 ms, maka akan mempengaruhi kinerja dari suatu jaringan. Berikut adalah cara untuk mendapatkan derajat keanggotaan berdasarkan data fuzzifikasi variabel *delay*.

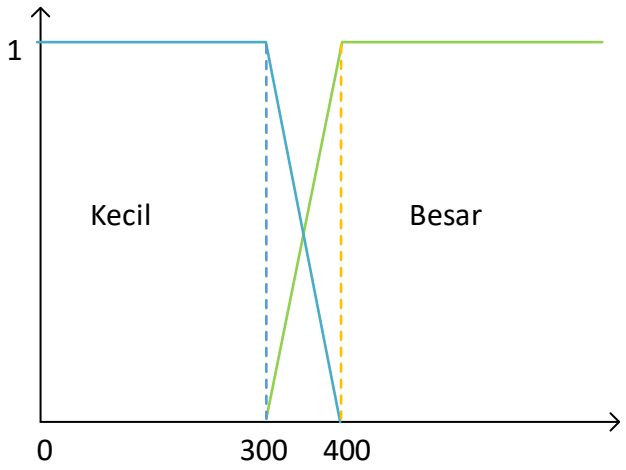
$$\mu[x]\ \text{Kecil} = \begin{cases} 1; & x \le 300 \\ \frac{400-x}{400-300}; & 300 \le x \le 400 \\ 0; & x \ge 400 \end{cases} \qquad (5.4)$$

$$\mu[x]\ \text{Besar} = \begin{cases} 0; & x \le 300 \\ \frac{x-300}{400-300}; & 300 \le x \le 400 \\ 1; & x \ge 400 \end{cases} \qquad (5.5)$$

Persamaan (5.4) digunakan utuk menghitung nilai derajat keanggotaan himpunan *fuzzy* variabel *delay* dengan kategori rendah, jika *delay* kurang dari 300, maka nilai derajat keanggotaannya akan bernilai 1 , kemudian jika *delay* lebih dari 300 dan kurang dari 400 maka nilai derajat keanggotaannya akan bernilai sesuai dengan hasil persamaan diatas, apabila *delay* lebih dari 400, maka nilai derajat keanggotaannya akan bernilai 0.

Persamaan (5.5) digunakan utuk menghitung nilai derajat keanggotaan himpunan *fuzzy* variabel *delay* dengan kategori rendah, jika *delay* kurang dari 300, maka nilai derajat keanggotaannya akan bernilai 0 , kemudian jika *delay* lebih dari 300 dan kurang dari 400 maka nilai derajat keanggotaannya akan bernilai sesuai dengan hasil persamaan diatas, apabila *delay* lebih dari 400, maka nilai derajat keanggotaannya akan bernilai 1.

Variabel *delay* dengan himpunan *fuzzy* kecil dan besar dapat direpresentasikan dalam sebuah grafik seperti pada gambar di bawah ini.



Gambar 5.9 Grafik derajat keanggotaan variabel *delay*

Pada gambar di atas terdapat 2 fungsi keanggotaan. Pertama adalah kategori kecil dengan bentuk grafik membership berupa trapesium bahu kiri. Kemudian yang kedua adalah kategori besar dengan bentuk grafik membership berupa trapesium bahu kanan. Variabel terakhir yang digunakan dalam logika *fuzzy* adalah variabel *output*.

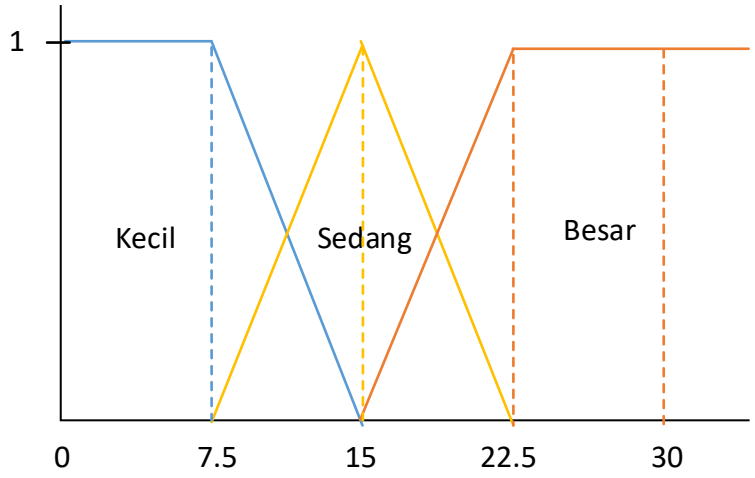
Variabel terakhir yang digunakan sebagai *output* adalah variabel bobot. Terdapat 3 himpunan *fuzzy* pada variabel bobot yakni kecil, sedang dan besar. Variabel ini nantinya akan digunakan sebagai keluaran dari aturan-aturan yang telah dibuat. Adapun hubungan linguistik dan numerik seperti pada tabel berikut.

Tabel 5.3 Fuzzifikasi variabel bobot

Linguistik	**Numerik**
Besar	15 - 30
Sedang	7.5 - 22.5
Kecil	0 - 15

Pada tabel diatas, terdapat 3 himpunan *fuzzy* yang digunakan yaitu kecil, sedang dan besar. Hal ini berdasarkan nilai range keseluruhan bobot yang bernilai 0 - 30 yang kemudian dibagi 3 sama rata dengan range 15 poin. Variabel bobot

dengan himpunan *fuzzy* kecil, sedang dan besar dapat di representasikan seperti pada gambar di bawah ini.



Gambar 5.10 Grafik derajat keanggotaan variabel bobot

Pada gambar di atas terdapat 3 fungsi keanggotaan. Pertama adalah himpunan *fuzzy* kecil dengan bentuk grafik membership berupa trapesium bahu kiri. Kemudian yang kedua adalah himpunan *fuzzy* sedang dengan bentuk grafik membership berupa segitiga. Terakhir adalah himpunan *fuzzy* besar dengan bentuk grafik membership berupa trapesium bahu kanan.

5.1.6.2 Rule-based *Fuzzy*

Rule-based merupakan suatu bentuk aturan implikasi if-then. Fungsi implikasi digunakan untuk menyusun baris aturan berupa implikasi *fuzzy* yang menyatakan relasi antara variabel *input* dan *output.* Terdapat 2 variabel *input* dan 1 variabel *input* yang digunakan dalam pembuatan aturan *fuzzy*. Dalam pembuatan aturan, akan digunakan “IF” dan “AND” dan akan menghasilkan keluaran “THEN”. Diharapkan jika kedua kondisi pada variabel *input* terpenuhi, maka akan menentukan *output* yang telah ditentukan. Aturan-aturan yang digunakan untuk menentukan kondisi *output* akan tersaji seperti pada tabel silang di bawah ini.

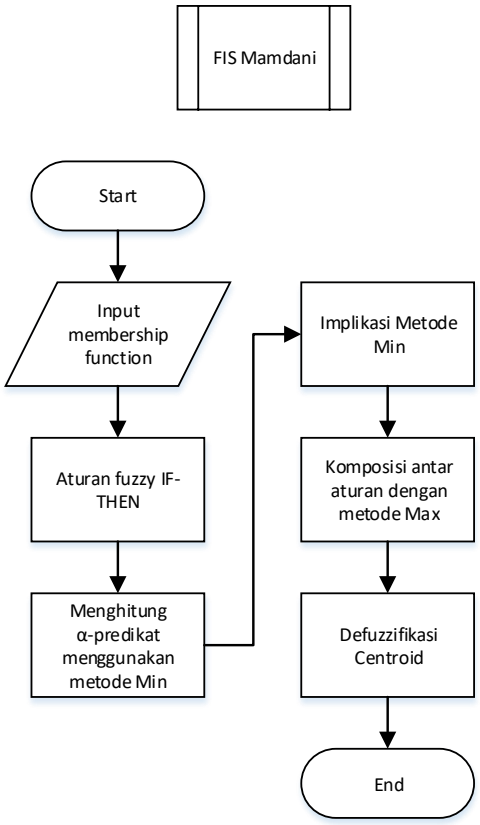
Tabel 5.4 Rule-based *Fuzzy*

Rule No.	*Traffic*	*Delay*	*Output* Bobot
1	Kecil	Kecil	Kecil
2	Kecil	Besar	Sedang
3	Sedang	Kecil	Sedang
4	Sedang	Besar	Besar
5	Besar	Kecil	Besar
6	Besar	Besar	Besar

Pada tabel di atas merupakan aturan-aturan yang dibuat dengan model IF-THEN berdasarkan himpunan *fuzzy* yang terdapat pada variabel *input*. Kemudian hasil dari aturan - aturan tersebut berupa himpunan *fuzzy* bobot yang akan menjadi dasar dari implikasi.

5.1.6.3 *Fuzzy Inference System Mamdani*

Fuzzy Inference System (FIS) Mamdani sering dikenal sebagai Metode Min-Max. Untuk mendapatkan *output* berupa *crisp value*, ada beberapa tahapan yang harus dilakukan seperti pada gambar di bawah ini.



Gambar 5.11 Flowchart FIS Mamdani

Tahap pertama adalah merubah variabel *input* dan variabel *output* menjadi beberapa himpunan *fuzzy* dan juga terdapat aturan-aturan *fuzzy* seperti yang sudah dibuat pada sub bab sebelumnya. Kemudian pada FIS Mamdani, fungsi implikasi yang digunakan adalah Min yang akan menghasilkan α-predikat. Setelah didapatkan hasil dari fungsi implikasi min, maka akan dilakukan komposisi antar aturan dengan metode Max yang akan menghasilkan daerah hasil *fuzzy*.

Berikut ini adalah contoh perhitungan FIS Mamdani. Dimisalkan variabel *traffic* bernilai 25 % dan variabel *delay* bernilai 5 ms. Setelah melalui proses fuzzifikasi, maka pada variabel *traffic* didapatkan nilai himpunan *fuzzy* kecil sebesar 0.75 , himpunan *fuzzy* sedang bernilai 0.25 dan himpunan *fuzzy* besar bernilai 0. Sedangkan pada variabel *delay* didapatkan nilai himpunan *fuzzy* kecil bernilai 1 dan himpunan *fuzzy* besar bernilai 0. Kemudian akan dilakukan proses inferensi seperti berikut ini.

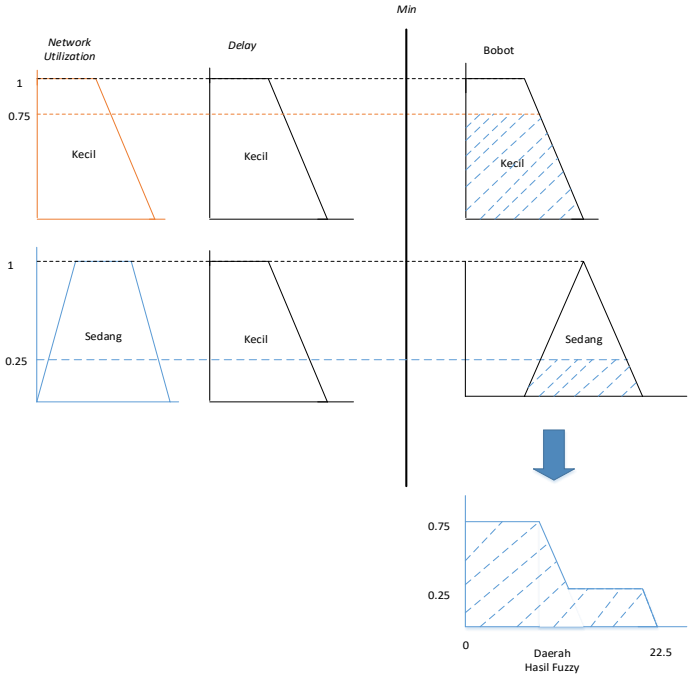
RULE[1]: IF *Traffic* Kecil AND *Delay* Kecil THEN Bobot Kecil

$$\begin{aligned} \alpha\text{-predikat}_1 &= min\ (\mu_{traffic}\ Kecil\ ,\ \mu_{Delay}\ Kecil\ ) \\ &= min\ (0.75\ ,\ 1) \\ &= 0.75 \end{aligned} \tag{5.6}$$

RULE[3]: IF *Traffic* Sedang AND *Delay* Kecil THEN Bobot Sedang

$$\begin{aligned} \alpha\text{-predikat}_3 &= min\ (\mu_{Traffic}\ Sedang\ ,\ \mu_{Delay}\ \text{Kecil}\ ) \\ &= min\ (0.25\ ,\ 1) \\ &= 0.25 \end{aligned} \tag{5.7}$$

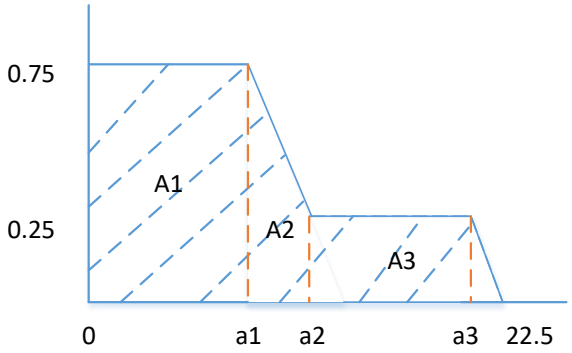
Kemudian setelah nilai α-predikat pada semua rule sudah diketahui, maka proses selanjutnya adalah melakukan implikasi dan komposisi antar aturan dengan menggunakan metode *Min-Max* sehingga akan menghasilkan daerah *fuzzy* baru seperti pada gambar di bawah ini.



Gambar 5.12 Metode *Min-Max*

Pada gambar di atas merupakan gambaran umum proses inferensi menggunakan metode Mamdani. Proses pertama diawali dengan mencari alfa predikat dari tiap aturan dengan menggunakan metode Min. Kemudian akan dilakukan implikasi dan komposisi antar aturan dengan menggunakan metode Min-Max yang selanjutnya akan menghasilkan daerah hasil *fuzzy* baru.

Setelah didapatkan daerah hasil *fuzzy*, maka selanjutnya adalah melakukakn defuzzifikasi. Pada penelitian ini, metode defuzzifikasi yang digunakan adalah metode centroid. Pada metode centroid, daerah hasil akan dibagi menjadi beberapa luas area. Untuk itu perlu dihitung batas-batas area terlebih dahulu. Berikut ini merupakan contoh batas luas area dan perhitungannya.



Gambar 5.13 Daerah hasil *fuzzy*

Pada metode centroid, daerah hasil *fuzzy* akan dibagi menjadi beberapa bagian. Pada gambar di atas daerah hasil *fuzzy* dibagi menjadi 3 bagian yaitu A1,

A2 dan A3. Kemudian terdapat batas-batas area yakni a1, a2 dan a3. Berikut merupakan contoh perhitungan untuk mencari nilai a1, a2 dan a3.

a1 = 15 - 0.75(15 - 7.5)
= 15 - 5.625 (5.8)
= 9.375

a2 = 15 - 0.25(15 - 7.5)
= 15 - 1.875 (5.9)
= 13.125

a3 = 22.5 - 0.25(22.5 - 15)
= 22.5 - 1.875 (5.10)
= 20.625

Dari persamaan di atas dapat ditentukan fungsi keanggotaan untuk daerah hasil seperti persamaan berikut ini.

$$\mu[z] = \begin{cases} 0.75; & z <= 9.375 \\ \frac{15-z}{15-7.5}; & 9.375 \le z \le 13.125 \\ 0.25; & 13.125 \le z \le 20.625 \\ \frac{22.5-z}{22.5-15}; & 20.625 \le z \le 22.5 \end{cases} \quad (5.11)$$

Dari persamaan di atas , untuk mendapatkan nilai *output* yang berupa *crisp value* dapat dihitung dengan menggunakan persamaan berikut ini.

$$Z^* = \frac{\sum_{j=1}^{n} Zj * \mu(Zj)}{\sum_{j=1}^{n} \mu(Zj)} \quad (5.12)$$

Berikut ini merupakan contoh perhitungan untuk merubah dari bilangan *fuzzy* menjadi *crisp value*.

$$Z^* = \frac{(3+6+9)*0.75 + (15+17+20)*0.25}{0.75*3 + 0.25*3}$$

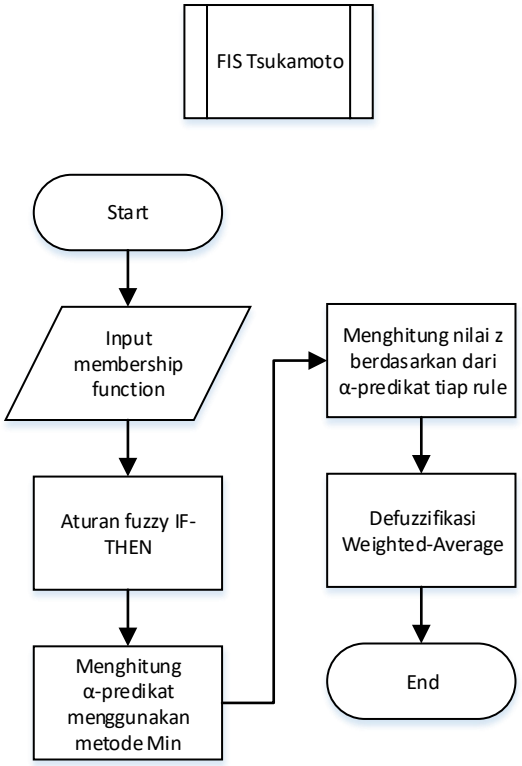
$$Z^* = \frac{26.5}{3} \quad (5.13)$$

$$Z^* = 8.83$$

Pada baris pertama pada perhitungan di atas , nilai 3, 6, 9, 15, 17 dan 20 merupakan nilai sebarang yang dipilih berdasarkan dengan nilai alfa predikat pada daerah hasil *fuzzy*. Dari perhitungan di atas dapat disimpulkan bahwa jika variabel *traffic* bernilai 25 % dan variabel *delay* bernilai 5 ms, maka akan menghasilkan nilai bobot *link* sebesar 8.83 .

5.1.6.4 *Fuzzy Inference System Tsukamoto*

Pada *Fuzzy Inference System* (FIS) *Tsukamoto*, setiap konsekuen pada aturan IF-THEN harus direpresentasikan dengan suatu himpunan *fuzzy* dengan fungsi keanggotaan yang monoton. Ada beberapa tahapan yang perlu dilakukan untuk mendapatkan nilai bobot berdasarkan rule-rule yang telah dibuat pada bagian sebelumnya, yaitu seperti pada gambar berikut ini.



Gambar 5.14 Flowchart *Fuzzy* Inference System Tsukamoto

Tahap pertama adalah merubah variabel *input* dan variabel *output* menjadi beberapa himpunan *fuzzy* dan juga terdapat aturan-aturan *fuzzy* seperti yang sudah dibuat pada sub bab sebelumnya. Kemudian tiap aturan akan dihitung nilai α-predikatnya dengan menggunakan metode Min. Dengan menggunakan permisalan yang sama dengan sebelumnya yakni jika nilai variabel *traffic* sebesar 25 % dan nilai variabel *delay* sebesar 5 ms ,maka contoh perhitungan α-predikat berdasarkan aturan-aturan yang telah dibuat seperti pada persamaan di bawah ini.

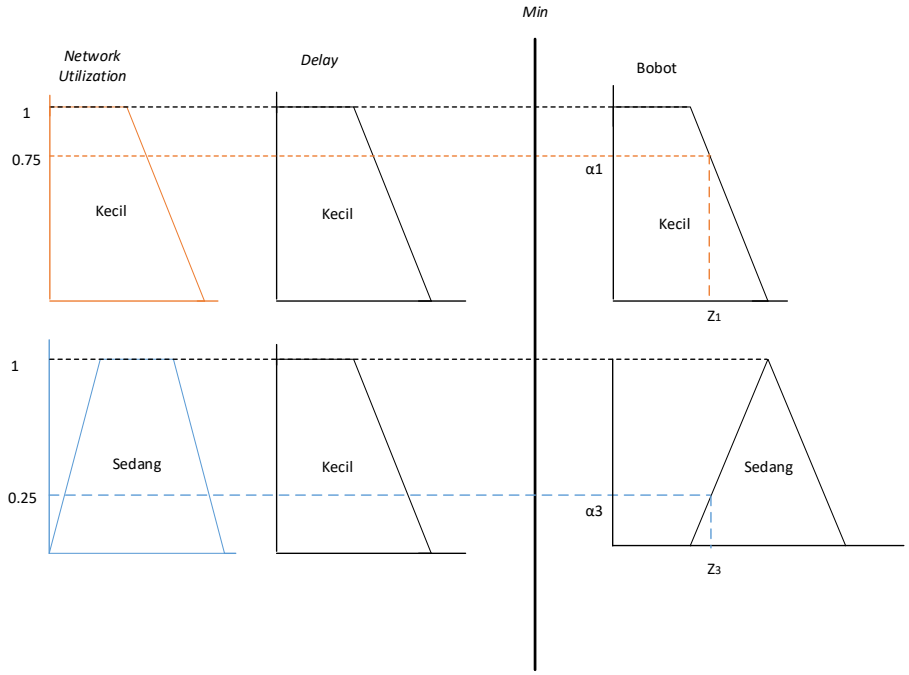
RULE[1]: IF *Traffic* Kecil AND *Delay* Kecil THEN Bobot Kecil

$$\begin{aligned} \alpha\text{-predikat}_1 &= min\ (\mu_{traffic}\ Kecil\ ,\ \mu_{Delay}\ Kecil\ ) \\ &= min\ (0.75\ ,\ 1) \\ &= 0.75 \end{aligned} \tag{5.14}$$

RULE[3]: IF *Traffic* Sedang AND *Delay* Kecil THEN Bobot Sedang

$$\begin{aligned} \alpha\text{-predikat}_3 &= min\ (\mu_{Traffic}\ Sedang\ ,\ \mu_{Delay}\ \text{Kecil}\ ) \\ &= min\ (0.25\ ,\ 1) \\ &= 0.25 \end{aligned} \tag{5.15}$$

Setelah didapatkan nilai α-predikat untuk setiap rule, maka langkah selanjutnya adalah menghitung *output* hasil inferensi (z) dari setiap aturan berdasarkan dengan α-predikatnya. Berikut ini merupakan gambar inferensi pada FIS Tsukamoto.



Gambar 5.15 *Fuzzy* Inference System Tsukamoto

Pada gambar di atas, hasil dari α-predikat akan digunakan dalam menentukan *output* hasil inferensi (z). Dikarenakan pada aturan yang dibuat menggunakan implikasi Min, maka bagian pada fungsi keanggotaan yang digunakan adalah sisi

sebelah kiri. Kemudian dari gambar di atas dapat dihitung nilai z berdasarkan dengan α-predikatnya. Berikut ini merupakan persamaan yang digunakan untuk menghitung nilai z.

$$z[\alpha]\ \text{Kecil} = \begin{cases} 0; & \alpha = 1 \\ 15 - \alpha(15 - 7.5); & 0 < \alpha < 1 \\ 15; & \alpha = 0 \end{cases} \quad (5.16)$$

$$z[\alpha]\ \text{Sedang} = \begin{cases} 15; & \alpha = 1 \\ \alpha(15 - 7.5) + 7.5; & 0 < \alpha < 1 \\ 7.5; & \alpha = 0 \end{cases} \quad (5.17)$$

$$z[\alpha]\ \text{Besar} = \begin{cases} 22.5; & \alpha = 1 \\ \alpha(22.5 - 15) + 15; & 60 \leq \alpha \leq 80 \\ 15; & \alpha = 0 \end{cases} \quad (5.18)$$

Pada persamaan di atas variabel z terdiri atas 3 himpunan *fuzzy* yaitu kecil, sedang dan besar. Persamaan di atas mengacu pada fungsi keanggotaan yang telah dibuat pada bagian fuzzifikasi sebelumnya. Kemudian dari persamaan di atas dapat dihitung nilai z untuk tiap aturan seperti pada contoh perhitungan berikut ini.

$$\begin{aligned} Z1\ [\text{Kecil}] &= 15 - \alpha 1(15 - 7.5) \\ &= 15 - 0.75(7.5) \\ &= 9.375 \end{aligned} \quad (5.19)$$

$$\begin{aligned} Z3\ [\text{Sedang}] &= \alpha 3(15 - 7.5) + 7.5 \\ &= 0.25(7.5) + 7.5 \\ &= 9.375 \end{aligned} \quad (5.20)$$

Setelah didapatkan nilai z untuk setiap aturan, maka langkah selanjutnya adalah melakukan defuzzifikasi. Defuzzifikasi dilakukan untuk merubah bilangan himpunan *fuzzy* menjadi *crisp value*. Pada FIS Tsukamoto, metode defuzzifikasi

yang digunakan adalah rata-rata terbobot (Weighted Average). Berikut ini merupakan persamaan yang digunakan dalam Weighted Average.

$$Z^* = \frac{\alpha pred1 * z1 + \alpha pred2 * z2 + \alpha pred3 \times z3 + \alpha pred4 * z4 + \cdots \alpha predi * zi}{\alpha pred1 + \alpha pred2 + \alpha pred3 + \alpha pred4 + \cdots \alpha predi} \quad (5.21)$$

αpredi = nilai alfa predikat ke-i

Zi = nilai Z ke-i

Dari persamaan di atas dapat dihitung nilai keluaran dari rule-rule yang telah dibuat. Berikut ini merupakan contoh perhitungan defuzzifikasi menggunakan metode Weighted Average.

$$Z^* = \frac{0.75 * 9.375 + 0 * 15 + 0.25 * 9.375 + 0 * 7.5 + 0 * 7.5 + 0 * 15 + 0 * 15}{0.75 + 0 + 0.25 + 0 + 0 + 0}$$

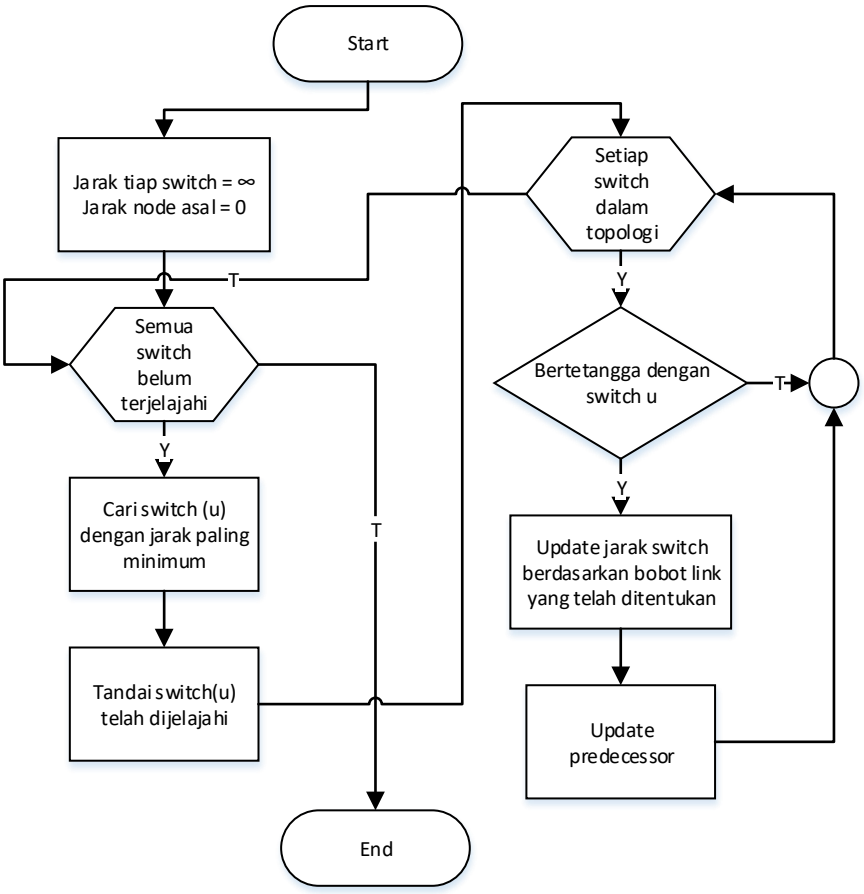
$$Z^* = \frac{9.375}{1} \quad (5.22)$$

$$Z^* = 9.375$$

Dari perhitungan di atas dapat disimpulkan bahwa jika variabel *traffic* bernilai 25 % dan variabel *delay* bernilai 5 ms, maka akan menghasilkan nilai bobot *link* sebesar 9.375.

5.1.7 Perancangan *Routing*

Pada perancangan *routing* berisi tentang bagaimana cara membangun sebuah fungsi *routing* yang dapat meneruskan paket dari pengirim menuju penerima dengan jalur yang telah ditentukan oleh algoritme *routing*. Algoritme *routing* yang digunakan adalah algoritme *Dijkstra*. Algoritme *Dijkstra* digunakan untuk mencari jalur terpendek ketika *host-2 (client)* melakukan request menuju *host*-1(*server).* Sistem akan melakukan pencarian jalur terpendek dengan bobot *link* terkecil pada topologi. Pada sistem ini satu host(*client*) hanya menggunakan satu jalur saja. Sistem akan melakukan *routing* kembali ketika ada host(client) lain yang melakukan request ke *server*. Diagram alir algoritme *Dijkstra* seperti pada gambar di bawah ini.



Gambar 5.16 Diagram Alir Algoritme *Dijkstra*

Pada gambar di atas , langkah pertama adalah menginisialisasi jarak ke semua switch menjadi tak terhingga dan memberi nilai jarak node asal menjadi 0. Kemudian akan mencari switch dengan jarak paling minimum yang akan ditandai agar tidak dijelajahi pada iterasi selanjutnya. Setelah switch tersebut didapatkan, langkah selanjutnya adalah melakukan update bobot dan predecessor pada switch yang bertetangga berdasarkan dengan bobot *link* yang telah ditentukan. Proses tersebut akan dilakukan sampai semua switch pada topologi berhasil dijelajahi. Hasil akhir pada algoritme ini yaitu berupa predecessor yang digunakan untuk mencari jalur dari switch asal menuju switch tujuan.

5.2 Implementasi Sistem

Berikut ini merupakan langkah-langkah yang dilakukan pada tahap implementasi berdasarkan dengan metodologi penelitian ada pada bagian sebelumnya.

5.2.1 Instalasi

Pada bagian ini memuat langkah-langkah yang dilakukan untuk memasang perangkat lunak pendukung untuk melakukan pengembangan sistem. Berikut ini merupakan langkah-langkah yang digunakan untuk pemasangan perangkat lunak pendukung.

5.2.1.1 Mininet

Mininet merupakan emulator jaringan yang digunakan untuk pengembangan pada sistem Software Defined Networking (SDN). Berikut ini dijelaskan bagaimana cara melakukan instalasi Mininet pada *Operating system Ubuntu*.

1. Mengunduh *source code Mininet* pada *github Mininet* dengan perintah berikut pada terminal:

   $ git clone git://github.com/mininet/mininet

2. Kemudian mulai proses installasi *Mininet* dengan perintah:

   $ sudo mininet/util/install.sh –a

5.2.1.2 Ryu Controller

Untuk melakukan instalasi *Ryu Controller* pada jaringan *OpenFlow*, Berikut ini merupakan langkah-langkah yang dapat dilakukan yaitu:

1. Melakukan instalasi *python-pip* terlebih dahulu dengan menjalankan perintah berikut pada terminal:

   $ sudo apt-get install python-pip

2. Untuk memulai proses instalasi *ryu* dengan perintah:

   $ sudo pip install ryu

5.2.1.3 *sFlow-RT*

Berikut ini merupakan prosedur untuk melakukan instalasi *sFlow-RT* yang digunakan sebagai *tools* untuk memantau *traffic* jaringan, yaitu sebagai berikut.

1. Mendownload *tarball* sFlow-RT dengan perintah:

   $ wget http://www.inmon.com/products/sFlow-RT/sflow-rt.tar.gz

2. Mengekstrak *tarball* sFlow-RT dengan perintah:

   $ tar xvzf sflow-rt.tar.gz

3. Menginstall *node.js* dengan perintah:

   $ sudo apt-get install node.js

5.2.1.4 Scikit-*Fuzzy*

Untuk melakukan instalasi *Scikit-Fuzzy* dapat dilakukan dengan mengikuti langkah-langkah berikut:

1. Melakukan instalasi *python-pip* terlebih dahulu dengan menjalankan perintah berikut pada terminal:

   $ sudo apt-get install python-pip

2. Melakukan instalasi NetworkX terlebih dahulu dengan menjalankan perintah berikut.

   $ sudo pip install networkx

3. Melakukan instalasi NumPy terlebih dahulu dengan menjalankan perintah berikut.

   $ sudo pip install numpy

4. Selanjutnya mulai proses instalasi *Scikit-Fuzzy* dengan perintah:

   $ sudo pip install scikit-*fuzzy*

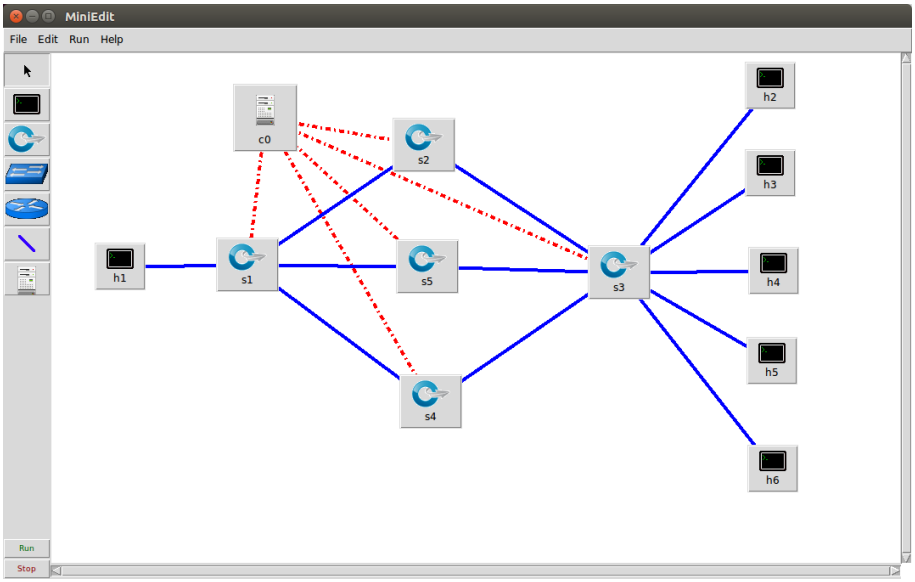
5.2.2 Membangun Topologi di Mininet

Setelah semua perangkat lunak pendukung telah terinstall, langkah selanjutnya adalah membangun topologi pada Mininet sesuai dengan desain topologi yang telah dilakukan. Salah satu cara untuk melakukan pembangunan topologi pada Mininet adalah dengan menggunakan GUI yang bernama Miniedit, yang akan dibahas di bagian ini. Berikut ini langkah-langkah membangun topologi di Miniedit, yaitu sebagai berikut.

1. Untuk Menjalankan *Miniedit* jalankah perintah berikut pada terminal:

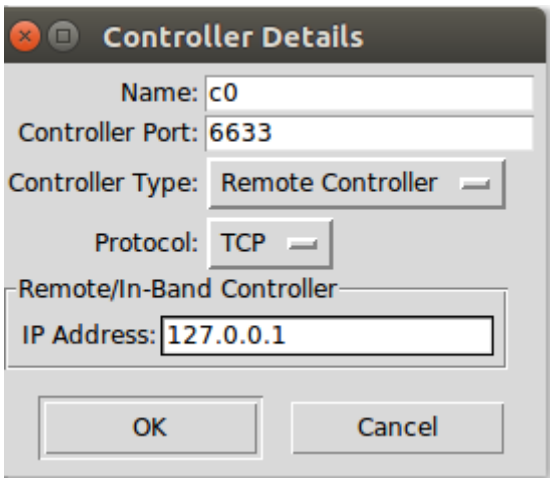
   $ sudo python mininet/examples/miniedit.py

2. Membangun topologi. Salah satu contoh hasil pembangunan topologi dapat dilihat pada gambar 5.14.



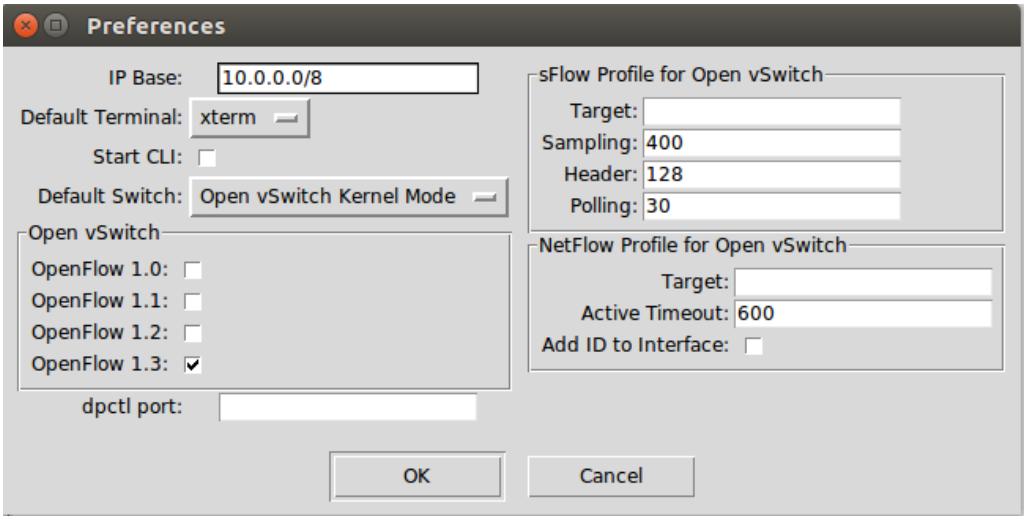
Gambar 5.17 Membangun Topologi pada Mininet

1. Melakukan pengaturan *controller* seperti gambar 4.6 dengan cara klik kanan pada logo *controller* (c0) kemudian memilih *properties*. Hal ini diperlukan agar nantinya *Mininet* dapat terhubung dengan *controller*. Pada *"Controller Type"* ubah menjadi "*Remote Controller*".



Gambar 5.18 Pengaturan Controller pada Mininet

3. Melakukan pengaturan pada *preferences* di *Miniedit* untuk mengaktifkan mengubah versi *OpenFlow* menjadi versi 1.3 seperti pada gambar 4.7. Hal ini dilakukan dengan cara memelih menu Edit kemudian memilih *Preferences*.



Gambar 5.19 Pengaturan *preferences* di *Mininet*

4. Untuk menjalankan simulasi dengan cara menekan tombol "*Run*".
5. Membuka terminal yang baru untuk menjalankan program *controller* dengan melakukan perintah berikut:

   $ sudo ryu-manager [nama_program].py --observe-*link*s

5.2.3 Pengembangan Program Controller

Pengembangan program *controller* terdiri atas langkah-langkah pemrograman yang dilakukan untuk *Ryu Controller* dengan menggunakan bahasa permrograman *Python* dalam mengimplementasi logika atau kecerdasan dari berdasarkan perancangan yang telah dilakukan.

5.2.3.1 *Source Code Monitoring Traffic*

Untuk dapat menghitung bobot *link*, dibutuhkan informasi tentang beban *traffic* pada suatu *link*. Informasi tersebut dapat diambil menggunakan tool *sFlow-RT* dengan melakukan request *HTTP GET*. Berikut ini implementasi dari fungsi *monitoring traffic* dengan sFlow-RT yang ditunjukkan pada kode berikut.

Source Code Monitoring Traffic	
1.	`def monitor_link(s1, s2):`
2.	`  while True:`
3.	`  tr1 = get_traffic(s1,s2)`
4.	`  tr2 = get_traffic(s2,s1)`
5.	`  traffic[s1.dpid][s2.dpid] = tr1 / b * 100`
6.	`  traffic[s2.dpid][s1.dpid] = tr2 / b * 100`
7.	`  time.sleep(0.5)`
8.	`def get_traffic(s1,s2):`

```
    link_traffic = 0
    url = 'http://' + collector + ':8008/metric/' + \
        collector + '/' +
str(switch_info[s1.dpid][s1.port_no]['ifindex']) + \
        '.ifoutoctets/json'
    r = get(url)
    response = r.json()
    try:
        link_traffic = response[0]['metricValue']*8
    except:
        link_traffic = 0
    return link_traffic
```

Pada *source code* di atas, terdapat fungsi "monitor_link" yang digunakan untuk *monitoring traffic* dengan input berupa pasangan switch yang bertetangga ("s1" dan "s2"). *Monitoring traffic* dilakukan setiap 0.5 detik untuk seluruh switch. Kemudian terdapat fungsi "get_traffic" yang digunakan untuk mengambil nilia traffic. Nilai *traffic* didapatkan dengan cara melakukan HTTP *request GET* pada REST API yang telah disediakan oleh sFlow-RT. Traffic yang dihasilkan oleh sFlow-RT berada dalam satuan *byte* sehingga perlu dikalikan 8 untuk mengubahnya menjadi satuan bit. Kemudian hasil dari pembacaan sFlow-RT akan dibandingkan dengan bandwidth ("b") sehingga akan menghasilkan *network utilization*.

5.2.3.2 Source Code Monitoring *Delay*

Untuk dapat menghitung bobot *link*, dibutuhkan informasi tentang *delay* yang ada pada suatu *link*. Informasi tersebut dapat diambil dengan memanfaatkan Library Ryu yaitu dengan cara mengirimkan paket ICMP ke semua switch yang bertetangga dengan dirinya. Berikut ini merupakan implementasi dari fungsi *monitoring delay* dengan Library Ryu yang ditunjukkan pada kode di bawah ini.

Source Code Monitoring Delay

```
def monitor_delay(self, s1,s2):
    while True:
        self.send_ping_packet(s1,s2)
        time.sleep(0.5)

def send_ping_packet(self, s1, s2):
    datapath = self.datapath_list[int(s1.dpid)]
    dst_mac = self.ping_mac
    dst_ip = self.ping_ip
```

```
10.     out_port = s1.port_no
11.     actions =
12. [datapath.ofproto_parser.OFPAction_Output_(out_port)
13.     pkt = packet.Packet()
14. pkt.add_protocol(ethernet.ethernet(ethertype=ether_ty
15. pes.ETH_TYPE_IP,src=self.controller_mac,dst=dst_mac))
16. pkt.add_protocol(ipv4.ipv4(proto=in_proto.IPPROTO_ICM
17. P,src=self.controller_ip,dst=dst_ip))
18.     echo_payload = '%s;%s;%f' % (s1.dpid, s2.dpid,
19. time.time())
20.     payload = icmp.echo(data=echo_payload)
21.     pkt.add_protocol(icmp.icmp(data=payload))
22.     pkt.serialize()
23.
24.     out = datapath.ofproto_parser.OFPPacketOut(
25.         datapath=datapath,
26.         buffer_id=datapath.ofproto.OFP_NO_BUFFER,
27.         data=pkt.data,
28.         in_port=datapath.ofproto.OFPP_CONTROLLER,
29.         actions=actions
30.     )
31.     datapath.send_msg(out)
32.
33. def ping_packet_handler(self, pkt):
34.     icmp_packet = pkt.get_protocol(icmp.icmp)
35.     echo_payload = icmp_packet.data
36.     payload = echo_payload.data
37.     info = payload.split(';')
38.     s1 = info[0]
39.     s2 = info[1]
40.     latency = (time.time() - float(info[2])) * 1000
41.     if latency > 1000:
42.         delay[int(s1)][int(s2)] = 1000
43.     else:
44.         delay[int(s1)][int(s2)] = int(latency)
45.
```

Pada kode di atas , terdapat fungsi “monitor_delay” yang digunakan untuk memonitoring *delay* dengan input berupa pasangan switch yang bertetangga (“s1” dan “s2”). Kemudian terdapat fungsi “send_ping_packet” yang digunakan untuk mendapatkan nilai delay dengan mengirimkan paket ICMP menuju switch yang

bertetangga. Paket ICMP yang dikirimkan memiliki payload berupa switch asal, switch tujuan dan waktu pengiriman. Paket tersebut akan dikirimkan menuju outport dari switch yang bertetangga. Kemudian terdapat fungsi "ping_packet_handler" yang digunakan untuk meng-handle paket ICMP tersebut. Ketika paket tersebut sampe pada switch tujuan , maka akan dihitung *delay* nya dengan cara waktu paket tiba dikurangi dengan waktu pengiriman paket dari switch asal. Kemudian hasil *delay* akan dikalikan dengan 1000 untuk merubah satuannya menjadi mili detik (ms).

5.2.3.4 Source Code Logika *Fuzzy* Mamdani

Setelah didapatkan nilai *traffic* dan *delay* , selanjutnya adalah menghitung nilai bobot menggunakan logika *fuzzy* Mamdani. Implementasi logika *fuzzy* Mamdani seperti pada kode berikut ini.

Source Code Logika Fuzzy Mamdani

```
1.  import numpy as np
2.  import skfuzzy as fuzz
3.  def Mamdani(param1,param2):
4.      tr  = np.arange(0,101,10)
5.      dl  = np.arange(0,1001,100)
6.      tr_kecil = fuzz.trapmf(tr, [0, 0, 20, 40])
7.      tr_sedang = fuzz.trapmf(tr, [20, 40, 60, 80])
8.      tr_besar = fuzz.trapmf(tr, [60, 80, 100, 100])
9.      dl_kecil = fuzz.trapmf(dl, [0, 0, 300, 400])
10.     dl_besar = fuzz.trapmf(dl, [300, 400, 1000, 1000])
11.
12.     w = np.arange(0,31,1)
13.     w_kecil  = fuzz.trapmf(w, [0, 0, 7.5, 15])
14.     w_sedang = fuzz.trimf(w, [7.5, 15, 22.5])
15.     w_besar = fuzz.trapmf(w, [15, 22.5, 30, 30])
16.
17.     def tr_category(tr_in):
18.         t_kecil =
19. fuzz.interp_membership(tr,tr_kecil,tr_in)
20.         t_sedang =
21. fuzz.interp_membership(tr,tr_sedang,tr_in)
22.         t_besar =
23. fuzz.interp_membership(tr,tr_besar,tr_in)
24.         return dict(kecil = t_kecil,sedang =
25. t_sedang, besar = t_besar)
```

```
26.
27.     def dl_category(dl_in):
28.         d_kecil = fuzz.interp_membership(dl,dl_kecil,
29.  dl_in)
30.         d_besar = fuzz.interp_membership(dl,dl_besar,
31.  dl_in)
32.         return dict(kecil = d_kecil, besar = d_besar)
33.
34.     tr_in = tr_category(param1)
35.     dl_in = dl_category(param2)
36.
37.     rule1 = np.fmin(tr_in['kecil'],dl_in['kecil'])
38.     rule2 = np.fmin(tr_in['kecil'],dl_in['besar'])
39.     rule3 = np.fmin(tr_in['sedang'],dl_in['kecil'])
40.     rule4 = np.fmin(tr_in['sedang'],dl_in['besar'])
41.     rule5 = np.fmin(tr_in['besar'],dl_in['kecil'])
42.     rule6 = np.fmin(tr_in['besar'],dl_in['besar'])
43.
44.     imp1 = np.fmin(rule1,w_kecil)
45.     imp2 = np.fmin(rule2,w_sedang)
46.     imp3 = np.fmin(rule3,w_sedang)
47.     imp4 = np.fmin(rule4,w_besar)
48.     imp5 = np.fmin(rule5,w_besar)
49.     imp6 = np.fmin(rule6,w_besar)
50.
51.     aggregate_membership = np.fmax(imp1,
52. np.fmax(imp2,np.fmax(imp3,np.fmax(imp4,np.fmax(imp5,i
53. mp6)))))
54.     result_w = fuzz.centroid(w, aggregate_membership)
55.     return result_w
```

Pada kode di atas menggunakan library pada python yaitu scikit-*fuzzy* dan numpy. Numpy digunakan untuk membuat variabel-variabel dasar berupa *array* yang digunakan dalam logika *fuzzy*. Variabel-variabel tersebut yaitu "tr" (*traffic*), "dl" (*delay*) dan "w" (bobot). Kemudian dengan menggunakan scikit-*fuzzy*, dapat melakukan fuzzifikasi dengan cara memanggil fungsi-fungsi keanggotaan yang ada library tersebut. Pada kode di atas menggunakan fungsi keanggotaan "trapmf" (trapesium) dan "trimf" (segitiga). Selanjutnya dengan menggunakan fungsi "interp_membership" , dapat mengubah *input* yang berupa *crisp value* menjadi derajat keanggotaan pada himpunan *fuzzy* yang telah dibuat. Kemudian akan dilakukan fungsi implikasi "fmin" (min) pada rule-rule dan himpunan *fuzzy output* yang telah dibuat. Selanjutnya adalah proses komposisi antar aturan dengan

menggunakan fungsi “fmax” (Max). Setelah didapatkan daerah hasil *fuzzy*, langkah selanjutnya yaitu melakukan defuzzifikasi dengan metode centroid dengan cara memanggil fungsi “centroid” sehingga menghasilkan *output* berupa *crisp value*.

5.2.3.5 Source Code Logika *Fuzzy* Tsukamoto

Setelah didapatkan nilai *traffic* dan *delay* , selanjutnya adalah menghitung nilai bobot menggunakan logika *fuzzy* Tsukamoto. Implementasi logika *fuzzy* Tsukamoto seperti pada kode berikut ini.

Source Code Logika Fuzzy Tsukamoto

```
1.  def Tsukamoto(param1,param2):
2.      rules = []
3.      zs = []
4.      tr_kecil = [0,0,20,40]
5.      tr_sedang = [20,40,60,80]
6.      tr_besar = [60,80,100,100]
7.      dl_kecil = [0,0,300,400]
8.      dl_besar = [300,400,1000,1000]
9.      w_kecil = [0,0,7.5,15]
10.     w_sedang = [7.5,15,22.5]
11.     w_besar = [15,22.5,30,30]
12.     t_in = int(param1)
13.     d_in = int(param2)
14.
15.     def interp_z_trap_turun(zmf,p):
16.         a = zmf[0]
17.         b = zmf[1]
18.         c = zmf[2]
19.         d = zmf[3]
20.         z = 0
21.         if p==1.0:
22.             z = b
23.         elif p > 0.0 and p < 1.0:
24.             z = d - p * (d - c)
25.         else:
26.             z = d
27.         return z
28.
29.     def interp_z_trap_naik(zmf,p):
30.         a = zmf[0]
```

```
        b = zmf[1]
        c = zmf[2]
        d = zmf[3]
        z = 0
        if p==1.0:
            z = b
        elif p > 0.0 and p < 1.0:
            z = p * (b - a) + a
        else:
            z = a
        return z

    def interp_z_tri(zmf,p):
        a = zmf[0]
        b = zmf[1]
        c = zmf[2]
        z = 0
        if p==1.0:
            z = b
        elif p > 0.0 and p < 1.0:
            z = p * (b - a) + a
        else:
            z = a
        return z

    def interp_trap(xmf,x):
        a = xmf[0]
        b = xmf[1]
        c = xmf[2]
        d = xmf[3]
        e = 0
        if x <= a and x >=d :
            e = 0
        elif x > a and x < b:
            e = (x - a) / float((b - a))
        elif x >= b and x <= c:
            e = 1
        elif x > c and x < d:
            e = (d - x) / float((d - c))
        else:
            e = 0
```

```
        return e

    def traffic_cat(traffic_in):
        traffic_level_lo =
interp_trap(tr_kecil,traffic_in)
        traffic_level_md =
interp_trap(tr_sedang,traffic_in)
        traffic_level_hi =
interp_trap(tr_besar,traffic_in)
        return dict(low = traffic_level_lo,medium =
traffic_level_md,high =
traffic_level_hi)

    def delay_cat(delay_in):
        delay_level_lo =
interp_trap(dl_kecil,delay_in)
        delay_level_hi =
interp_trap(dl_besar,delay_in)
        return dict(low = delay_level_lo, high =
delay_level_hi)

    traffic_in = traffic_cat(t_in)
    delay_in = delay_cat(d_in)

    rule1 = min(traffic_in['low'],delay_in['low'])
    rule2 = min(traffic_in['low'],delay_in['high'])
    rule3 =
min(traffic_in['medium'],delay_in['low'])
    rule4 =
min(traffic_in['medium'],delay_in['high'])
    rule5 = min(traffic_in['high'],delay_in['low'])
    rule6 =
min(traffic_in['high'],delay_in['high'])

    z1 = interp_z_trap_turun(w_kecil,rule1)
    z2 = interp_z_tri(w_sedang,rule2)
    z3 = interp_z_tri(w_sedang,rule3)
    z4 = interp_z_trap_naik(w_besar,rule4)
    z5 = interp_z_trap_naik(w_besar,rule5)
    z6 = interp_z_trap_naik(w_besar,rule6)

```

```
113.     for i in range(1,7):
114.         a = locals()['rule%s' % i]
115.         b = locals()['z%s' % i]
116.         rules.append(a)
117.         zs.append(b)
118.
119.     atas = sum([a * b for (a, b) in zip(rules,
120.  zs)])
121.     bawah = sum(rules)
122.     tsk = atas/float(bawah)
123.     return tsk
```

Pada kode di atas menggunakan built-in library pada python. Terdapat beberapa variabel yang digunakan sebagai *input* dan *output*. Variabel tersebut yaitu "tr" (*traffic*) dan "dl" (*delay*) sebagai *input* kemudian variabel "w" (bobot) sebagai *output*. Masing-masing variabel tersebut dibagi menjadi beberapa himpunan *fuzzy* yaitu kecil, sedang dan besar. Kemudian terdapat beberapa method yang berfungsi sebagai membership function pada logika *fuzzy*. Pada kode di atas menggunakan fungsi keanggotaan trapesium "interp_trap" dan "interp_z_x" yang digunakan untuk mencari nilai Z. Selanjutnya dengan menggunakan fungsi keanggotaan tersebut dapat mengubah *input* yang berupa *crisp value* menjadi derajat keanggotaan pada himpunan *fuzzy* yang telah dibuat. Kemudian akan dilakukan fungsi "min" sehingga menghasilkan α-predikat untuk setiap rule. Selanjutnya adalah mencari nilai "z" menggunakan method "interp_z_x" berdasarkan dengan nilai α-predikat nya. Setelah didapatkan nilai α-predikat dan z untuk setiap rule, langkah selanjutnya yaitu melakukan defuzzifikasi dengan metode weighted average dengan cara melakukan perhitungan seperti pada persamaan 5.21 sehingga menghasilkan *output* berupa *crisp value*.

5.2.3.3 Source Code Routing Algoritme Dijkstra

Setelah didapatkan nilai bobot menggunakan logika *fuzzy*, selanjutnya adalah pencarian jalur terpendek menggunakan algoritme *Dijkstra*. Jalur akan dipilih berdasarkan *link* yang memiliki bobot paling kecil. Berikut ini merupakan implementasi algoritme *Dijkstra*.

Source Code Algoritme Dijkstra

```
1.  def minimum_distance(distance, Q):
2.      min = float('Inf')
3.      node = 0
4.      for v in Q:
5.          if distance[v] < min:
6.              min = distance[v]
7.              node = v
8.      return node
9.
10. def get_path(src, dst, first_port, final_port):
11.     distance = defaultdict(lambda: float('Inf'))
12.     previous = defaultdict(lambda: None)
13.     distance[src] = 0
14.     Q = set(switches)
15.     bobot = defaultdict(lambda: defaultdict(lambda:
16. None))
17.     while len(Q) > 0:
18.         u = minimum_distance(distance, Q)
19.         Q.remove(u)
20.         for p in switches:
21.             if adjacency[u][p] != None:
22.                 w = 1
23.                 #w =
24. Tsukamoto(traffic[u][p],delay[u][p])
25.                 #w =
26. Mamdani(traffic[u][p],delay[u][p])
27.                 bobot[u][p] =
28. (w,traffic[u][p],delay[u][p])
29.                 if distance[u] + w < distance[p]:
30.                     distance[p] = distance[u] + w
31.                     previous[p] = u
32.     return previous
```

Pada kode di atas terdapat fungsi "*minimum_distance"* yang digunakan untuk mencari jarak switch paling minimum. Kemudian terdapat fungsi "get_path" yang digunakan untuk pencarian jalur. Pencarian jalur terpendek di mulai dengan memberi nilai 0 pada swith asal dan nilai tak terhingga untuk switch yg lain. Kemudian akan mencari switch(u) dengan jarak paling minimum menggunakan fungsi "minimum_distance". Selanjutnya adalah melakukan pencarian jalur. Pencarian jalur dilakukan dengan cara memperbarui jarak switch berdasarkan

bobot yang telah ditentukan. Pada penelitian ini menggunakan 3 variasi bobot yang berbeda. Bobot yang pertama yaitu bernilai 1 (“w = 1”) yang digunakan pada algoritme *Static Cost Dijkstra*. Kemudian bobot yang ke-2 bernilai “Tsukamoto(*traffic*[u][p],*delay*[u][p])” dimana bobot tersebut dihasilkan dari perhitungan logika *fuzzy* Tsukamoto. Kemudian variasi bobot yang ke-3 adalah “Mamdani(*traffic*[u][p],*delay*[u][p])” dimana bobot tersebut hasil dari perhitungan menggunakan logika *fuzzy* Mamdani. Hasil dari pencarian algoritme Dijkstra tersebut berupa variabel “*previous”* yang berisi *predecessor* untuk setiap switch yang ada pada topologi. Kemudian dari *predecessor* tersebut dapat dibentuk jalur dari *switch* asal menuju *switch* tujuan.