BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Pada bagian ini dilakukan kajian terhadap penelitian-penelitian sebelumnya yang terkait dengan penelitian ini dan dijadikan sebagai pedoman dalam pelaksanaan penelitian. Berikut tabel perbandingan penelitian terdahulu dan sekarang:

Tabel 2.1 Kajian Pustaka

No	Nama Penulis, Tahun dan Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	**Rencana Penelitian**
1	Lazuardi, Ericko. [2016]. *Metode Pemilihan Jalur Routing Adaptif dengan Menggunakan Algoritme Dijkstra pada Openflow Network*	Mengimplementasikan algoritme *Dijkstra* pada OpenFlow SDN	Routing menggunakan algoritme *Dijkstra* pada SDN dengan mempertimbangkan kepadatan jaringan dengan cara memberi threshold pada *traffic* rate.	Mempertimbangkan *traffic* dan *delay* pada SDN dengan penentuan bobot *link* secara dinamis menggunakan logika *fuzzy*.
2	Prayogi, Agus. [2017]. Sistem Pendukung Keputusan Untuk Penentuan Jumlah Produksi Nanas Menggunakan Metode *Fuzzy Tsukamoto*	Menggunakan logika *fuzzy tsukamoto*	Menentukan jumlah produksi nanas berdasarkan persediaan dan jumlah permintaan menggunakan *fuzzy Tsukamoto*	Menentukan bobot *link* pada algoritme *Dijkstra* dengan menggunakan logika *fuzzy Tsukamoto*

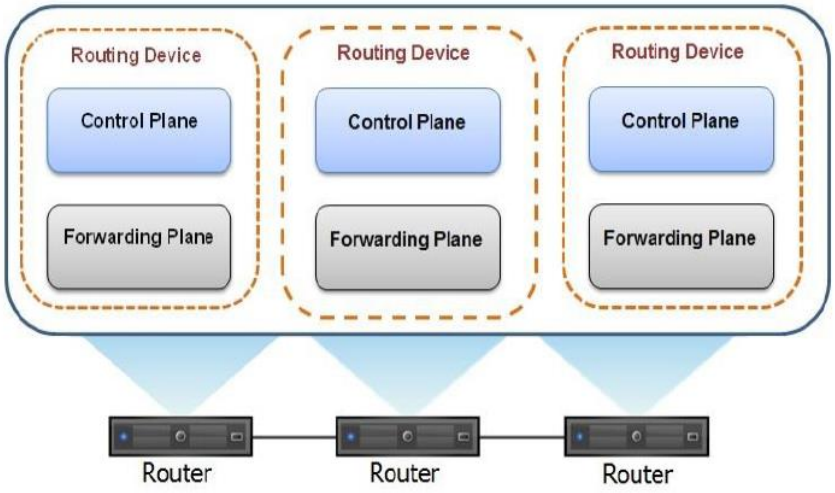
3	Septiawan, Reino Adi.[2013]. IMPLEMENTASI LOGIKA FUZZY MAMDANI UNTUK MENENTUKAN HARGA GABAH	Menggunakan logika *fuzzy Mamdani*	Menentukan harga gabah berdasarkan kadar air dan kotoran menggunakan logika *fuzzy Mamdani*	Menentukan bobot *link* pada algoritme *Dijkstra* dengan menggunakan logika *fuzzy Mamdani*

2.2 Dasar Teori

Pada subbab ini, dijabarkan berbagai teori-teori yang dapat mendukung penelitian ini, yaitu berbagai teori tentang algoritme *routing* dan logika *fuzzy* pada *Software-Defined Networking*.

2.2.1 Software-Defined Networking

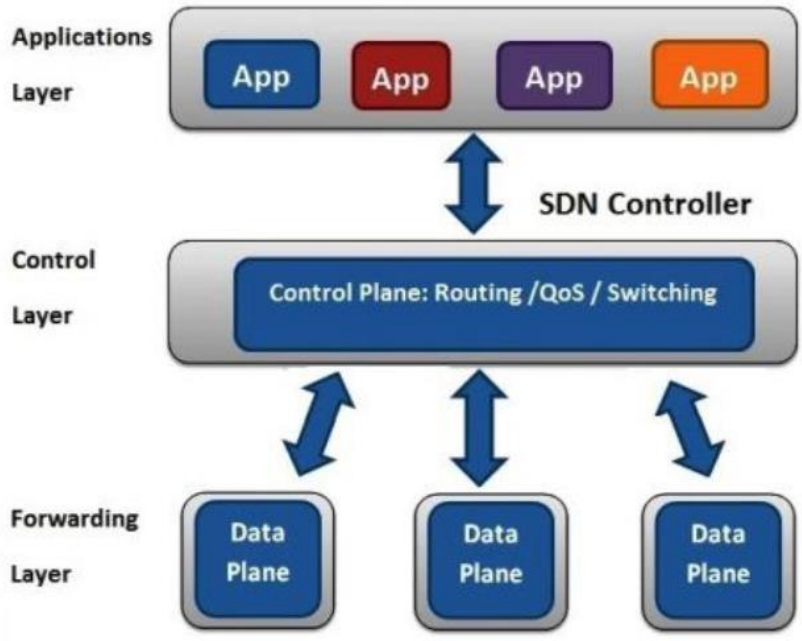
Software-Defined Networking (SDN) merupakan arsitektur yang dinamis, mudah dikelola, *cost-efficient*, dan mudah beradaptasi, sehingga ideal untuk sistem yang memiliki *bandwidth* tinggi (Foundation, 2017). Arsitektur ini memisahkan antara *control plane* dan *forwarding plane* yang memungkinkan mengontrol jaringan secara langsung dan terpusat. *Control plane* merupakan bagian yang berfungsi untuk mengatur logika di dalam jaringan seperti algoritme *routing* dan tabel *routing*, sedangkan *data plane* atau *forwarding plane* adalah bagian yang berfungsi untuk mengatur *forwarding* (Sonba & Abdalkreim, 2014). Pada arsitektur jaringan konvensional, *control plane* dan *data plane* berada pada perangkat yang sama seperti pada Gambar 2.1 berikut.



Gambar 2.1 Arsitektur Jaringan Konvensional

Sumber : (Sonba & Abdalkreim, 2014)

Pada jaringan konvensional, *control plane* dan *data plane* berada pada perangkat keras yang sama, hal ini akan membuat meningkatnya kompleksitas pada jaringan seiring dengan bertambahnya kebutuhan akan pertukaran informasi pada jaringan. Dengan pemisahan *control plane* dan *data plane,* SDN diharapkan dapat mengatasi kompleksitas yang ada dalam jaringan traditional. Untuk arsitektur SDN dapat dilihat pada Gambar 2.2 berikut ini.



Gambar 2.2 Arsitektur Jaringan pada SDN

Sumber : (Sonba & Abdalkreim, 2014)

Pada arsitektur SDN terbagi menjadi 3 layer yaitu layer aplikasi, layer *control* dan layer *forwarding.* Layer aplikasi adalah aplikasi yang berjalan pada jaringan seperti *web server, mail server* dan lain-lain. Layer kontrol adalah layer yang berfungsi sebagai entitas kontrol atau SDN *Controller* mentranlasikan kebutuhan aplikasi dengan memberikan instruksi yang sesuai untuk SDN *datapath* serta memberikan informasi yang relevan dan dibutuhkan oleh SDN *Application.* Dan yang ketiga layer forwarding adalah sekumpulan perangkat jaringan yang akan dikendalikan oleh layer kontrol. Dengan memisahkan *control plane* dan *data plane,* SDN menawarkan pemeliharaan, pembuatan dan mendisain jaringan yang lebih inovatif dan fleksibel.

2.2.2 Ryu

Ryu merupakan sebuah *framework* berbasis komponen untuk SDN. *Ryu* menyediakan komponen *software* dengan API yang memudahkan *developer* untuk membangun aplikasi-aplikasi kontrol dan manajemen jaringan (Ryu SDN Framework Community, 2014). Pengembangan aplikasi untuk *Ryu* dapat dilakukan dengan menggunakan bahasa *Python* atau dengan mengirimkan pesan JSON

melalui API yang tersedia. *Ryu* mendukung berbagai protokol untuk memanajemen jaringan antara lain *OpenFlow, NetConf, Of-config* dll. Kebutuhan atas *Ryu* sebagai *OpenFlow controller* adalah karena *Ryu* mendukung *OpenFlow* versi 1.0 hingga 1.5. *Ryu* di sini digunakan sebagai *Controller* SDN dan sebagai *framework* untuk mengembangkan sebuah sistem *routing* di *OpenFlow* SDN.

2.2.3 Mininet

Dibanding dengan membeli komponen SDN yang nyata, *Mininet* memberikan penawaran yang lebih mudah dan murah dalam penelitian dibidang SDN. Mininet adalah sebuah emulator yang memungkinkan membuat jaringan virtual seperti *host, switch*, maupun *controller*. *Mininet* memungkinkan untuk membuat disain dan menjalankan prototype SDN. Dengan menggunakan Mininet dapat membuat topologi yang diinginkan, seberapa besar *bandwidth* pada tiap *link* (Mininet Team, 2016). Dalam Mininet terdapat beberapa topologi bawaan seperti pada Tabel 2.4 berikut.

Tabel 2.2 *Command Default* Topologi *Mininet*

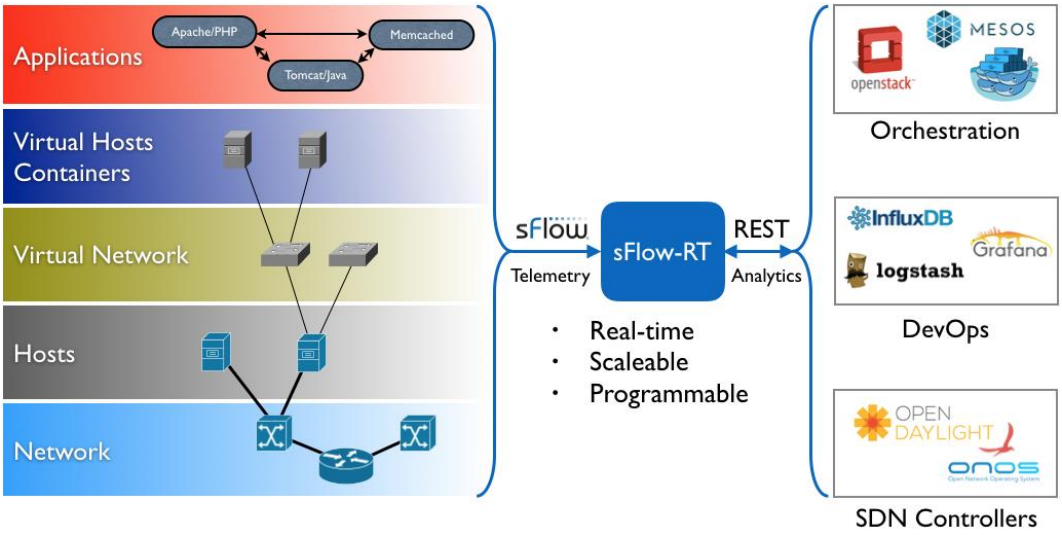
Sumber : (Mininet Team, 2016)

Command	Keterangan
$sudo mn --topo tree,3 --mac –switch ovsk --controller=remote	membuat single topologi
$sudo mn --topo linear,3 --mac --switch ovsk --controller=remote	membuat tree topologi
$sudo mn --topo single,3 --mac --switch ovsk --controller=remote	membuat linear topologi

Selain dapat membuat kustom topologi menggunakan bahasa Python, *Mininet* juga menyediakan tampilan berbasis GUI sehingga mempermudah membuat topologi yang diinginkan. *Mininet* juga memungkinkan menggunakan program *client-server* seperti perintah *ping*, *iperf*, *ifconfig*, *wget*, dan lain-lain.

2.2.4 sFlow-RT

sFlow-RT merupakan sebuah *tool* untuk memonitor jaringan secara *real-time* untuk SDN dengan menggunakan teknologi analitik asinkron dari InMon dan memungkinkan aplikasi pada jaringan SDN yang memperhatikan performa, seperti *load-balancing*, perlindungan terhadap DDoS, dan sebagainya (InMon, 2017).



Gambar 2.3 sFlow-RT

Sumber: (InMon, 2017)

sFlow-RT akan menerima aliran telemetri secara kontinu dari agen-agen *sFlow* yang terpasang pada perangkat-perangkat jaringan, *host*, dan aplikasi dan mengubahnya menjadi metrik yang dapat ditindaklanjuti dan dapat diakses pada sebuah REST API. REST API tersebut memudahkan berbagai hal pada SDN antara lain melakukan konfigurasi terhadap pengukuran yang terkostumisasi, menerima metrik, menetapkan *threshold*, dan menerima notifikasi. Aplikasi terhadap sFlow dapat dibuat secara eksternal yang dapat ditulis dalam bahasa apapun yang mendukung pemanggilan HTTP/REST, ataupun secara internal dalam sFlow-RT dengan menggunakan bahasa JavaScript/ECMAScript. (InMon, 2017).

2.2.5 Routing

Routing adalah proses menentukan rute atau jalur yang diambil oleh paket dimana mereka mengalir dari pengirim ke penerima. Setiap *router* memiliki tabel *forwarding*. Sebuah router meneruskan paket dengan memeriksa nilai *field* dalam *header* saat paket tiba, dan kemudian menggunakan *header* ini sebagai nilai indeks ke tabel *forwarding router*. Nilai yang tersimpan dalam *forwarding* tabel menunjukkan *outport router* untuk paket yang akan diteruskan. Algoritme yang menghitung jalur ini disebut algoritme *routing*. Sebuah algoritme *routing* akan menentukan, Misalnya, jalan di sepanjang dimana paket mengalir dari H1 ke H2. Algoritme *routing* menentukan nilai-nilai yang dimasukkan ke dalam tabel *forwarding* router. Algoritme *routing* dapat terpusat (misalnya, dengan algoritme mengeksekusi di situs pusat dan download informasi *routing* untuk masing-masing router) atau desentralisasi (yaitu, dengan sepotong algoritme routing terdistribusi berjalan di setiap router). Salah satu algoritme routing yang terkenal adalah algoritme *Link-State* atau yang biasa disebut dengan algoritme *Dijkstra*. Dalam algoritme *link*-state, topologi jaringan dan bobot semua *link* diketahui oleh setiap node, topologi dan bobot *link* tersebut kemudian akan diproses sebagai masukan

untuk algoritme LS. Dalam prakteknya, hal ini dicapai dengan setiap node melakukan broadcast yang berisi paket *Link-State* dimana paket tersebut berisi bobot *link* yang terhubung dan identitas masing-masing node. Protokol *routing* yang menggunakan algoritme *Dijkstra* adalah *Open Shortest Path First* (OSPF). Pada OSPF, bobot dari setiap *link* akan dikonfigurasi oleh administrator jaringan. Administrator dapat memilih untuk mengatur semua biaya *link* ke 1, sehingga mencapai *minimum-hop routing*, atau mungkin memilih untuk mengatur bobot *link* proporsional dengan ukuran *bandwidth* (Kurose & Ross, 2013).

2.2.6 Algoritme Dijkstra

Dalam algoritme *Dijkstra* atau yang biasa disebut *link-state*, topologi jaringan dan bobot semua *link* diketahui oleh setiap *node*, topologi dan bobot *link* tersebut kemudian akan diproses sebagai masukan untuk algoritme LS. Dalam prakteknya, hal ini dicapai dengan setiap *node* melakukan broadcast yang berisi paket *Link-State* dimana paket tersebut berisi bobot *link* yang terhubung dan identitas masing-masing *node*. Algoritme *Dijkstra* menghitung jalur dengan bobot paling kecil dari satu *node* (sumber, yang disebut sebagai u) ke semua *node* lainnya dalam jaringan. Algoritme *Dijkstra* melakukan iterasi dan memiliki properti bahwa setelah iterasi k, jalur paling sedikit bobotnya akan diketahui ke *node* tujuan, dan dibandingkan jalur lain ke semua *node* tujuan, k ini jalur yang akan memiliki biaya terkecil. (Kurose & Ross, 2013)

Kode Listing 2.1 Pseudokode Algoritme Dijkstra

Sumber : (Kurose & Ross, 2013)

```
1.  Initialization:
2.      N' = {u}
3.      for all nodes v
4.          if v is a neighbor of u
5.              then D(v) = c(u,v)
6.          else D(v) = ∞
7.  Loop
8.      find w not in N' such that D(w) is a minimum
9.      add w to N'
10.     update D(v) for each neighbor v of w and not in N':
11.         D(v) = min( D(v), D(w) + c(w,v) )
12.     /* new cost to v is either old cost to v or known
13.     least path cost to w plus cost from w to v */
14. until N'= N
```

Node u merupakan titik awal pencarian jalur terpendek ke tujuan. Untuk setiap *node* yang bertetangga dengan *node* u, maka D(v) adalah bobot dari *node* u ke *node* v. Jika tidak bertetangga maka D(v) akan bernilai tak terhingga. Kemudian

akan melakukan iterasi untuk mencari jalur terpendek. Cari bobot paling kecil (D(w)) dari tetangga yang sudah diketahui bobotnya. Kemudian tambahkan *node* w ke dalam N'. Setelah itu D(v) akan memperbarui bobotnya dengan mempertimbangkan *node* w. Algoritme ini akan melakukan iterasi sampai menemukan jalur terpendek dari *node* asal ke tujuan.

2.2.7 Throughput

Throughput merupakan nilai rata-rata paket yang tiba melalui saluran transmisi (Bonaventure, 2011). *Throughput* dapat diukur dalam satuan bit per detik (bps) atau paket per detik (pps). *Throughput* juga mengukur efisiensi dan efektivitas kinerja protokol dan menentukan kinerja jaringan dari satu *node* ke tujuan. *Tools* yang digunakan untuk menguji *throughput* adalah *iperf*. Sintaks yang digunakan pada *iperf* adalah sebagai berikut :

-b Data format

-r Bi-directional bandwidth

-d Simultaneous bi-directional bandwidth

-w TCP windows size

-p port

-t timing

-i interval

-u, -b UDP tests, bandwidth settings

-m maximum segments size display

-M maximum segments size settings

-P parallel tests

-h help

Contoh penggunaan *iperf* : *iperf* –c 10.0.0.1 –t 10 –i 1, pada contoh ini adalah bertindak sebagai klien dengan mengirimkan paket *iperf* menuju *server* dengan alamat 10.0.0.1 dalam waktu 10 detik dan paket dikirimkan setiap 1 detik.

2.2.8 Latency

Latency adalah waktu yang dibutuhkan sebuah paket untuk ditransmisikan dari satu *node* ke *node* yang lainnya. *Latency* dapat dipengaruhi oleh media fisik, jarak, kemacetan jaringan atau juga waktu proses yang lama. Untuk pengujian *latency*, dapat dilakukan dengan mengirim paket ICMP kepada tujuan seperti berikut ini (Tanenbaum & Wetherall, 2011).

#ping –s 20000 –c 10 10.0.0.1

-s 20000 ,ukuran paket adalah 20000 byte

-c 10, paket dikirim sebanyak 10 kali

-10.0.0.1 adalah alamat IP server

2.2.9 Packet Loss

Packet Loss terjadi ketika satu atau beberapa paket data yang melintasi suatu jaringan komputer gagal mencapai tujuannya. *Packet Loss* biasanya disebabkan oleh *traffic* yang padat pada jaringan. Packet loss diukur sebagai persentase paket yang hilang sehubungan dengan paket yang dikirim (Wikipedia, 2017).

2.2.10 Convergence Time

Convergence merupakan sebuah proses pada router dimana router tersebut menyetujui rute optimal untuk meneruskan paket dan memperbarui routing tabel. *Convergence Time* yang optimal adalah yang memiliki waktu yang kecil (Project, 2005).

2.2.11 Propagation Delay

Delay propagasi adalah istilah teknis yang bisa memiliki arti berbeda tergantung konteksnya. Hal ini dapat berhubungan dengan jaringan komputer, elektronika atau fisika. Secara umum *delay* propagasi adalah lamanya waktu yang dibutuhkan mencapai tujuannya. Dalam jaringan komputer, *delay* propagasi adalah jumlah waktu yang dibutuhkan bagi sinyal untuk melakukan perjalanan dari pengirim ke penerima. Hal ini dapat dihitung sebagai rasio antara panjang *link* dan kecepatan propagasi pada medium tertentu (Wikipedia, 2017).

2.2.12 Cumulative Distribution Function (CDF)

Cumulative Distribution Function (CDF) dari variabel acak adalah metode lain untuk menggambarkan distribusi variabel acak. Keuntungan dari CDF adalah dapat didefinisikan untuk jenis variabel acak (diskrit, kontinu, dan campuran) (Pishro-Nik, 2017).

2.2.13 Logika *Fuzzy*

Konsep tentang logika *fuzzy* diperkenalkan oleh Prof. Lotfi Astor Zadeh pada 1962. Logika *fuzzy* adalah metodologi sistem kontrol pemecahan masalah, yang cocok untuk diimplementasikan pada sistem, mulai dari sistem yang sederhana, sistem kecil, *embedded system*, jaringan PC, *multichannel* atau *workstation* berbasis akuisisi data, dan sistem kontrol. Metodologi ini dapat diterapkan pada perangkat keras, perangkat lunak, atau kombinasi keduanya. Dalam logika klasik dinyatakan bahwa segala sesuatu bersifat biner, yang artinya adalah hanya mempunyai dua kemungkinan, “Ya atau Tidak”, “Benar atau Salah”,

“Baik atau Buruk”, dan lain-lain. Oleh karena itu, semua ini dapat mempunyai nilai keanggotaan 0 atau 1. Akan tetapi, dalam logika *fuzzy* kemungkinan nilai keanggotaan berada diantara 0 dan 1. Artinya,bisa saja suatu keadaan mempunyai dua nilai “Ya dan Tidak”, “Benar dan Salah”, “Baik dan Buruk” secara bersamaan, namun besar nilainya tergantung pada bobot keanggotaan yang dimilikinya (Sutojo, 2011).

2.2.13.1 Fungsi Keanggotaan

Fungsi keanggotaan adalah grafik yang mewakili besar dari derajat keanggotaan masing-masing variabel *input* yang berada dalam interval antara 0 dan 1. Derajat keanggotaan sebuah variabel *x* dilambangkan dengan simbol μ(*x*). R*ule-rule* menggunakan nilai keanggotaan sebagai faktor bobot untuk menentukan pengaruhnya pada saat melakukan inferensi untuk menarik kesimpulan (Sutojo, 2011). Ada beberapa fungsi yang bisa digunakan antara lain :

1. Representasi Linear, pada representasi linear pemetaan *input* ke derajat keanggotaannya digambarkan sebagai suatu garis lurus. Bentuk ini paling sederhana dan menjadi pilihan yang baik untuk mendekati suatu konsepyang kurang jelas. Ada dua keadaan *fuzzy* yang linear yaitu representasi linear naik dan representasi linear turun.
2. Representasi Kurva Segitiga, Kurva segitiga pada dasarnya merupakan gabungan antara dua garis linear.
3. Representasi Kurva Trapesium, Kurva trapesium pada dasarnya seperti bentuk segitiga, hanya saja ada beberapa titik yang memiliki nilai keanggotaan 1 (Kusumadewi, 2003).

2.2.13.2 Fuzzy Inference System

Sebuah *Fuzzy Inference System* (FIS) adalah cara variabel ruang *input* pemetaan untuk satu atau lebih variabel ruang *output* menggunakan logika *fuzzy* (Ross, 2010). Sebuah FIS terdiri dari beberapa komponen, yaitu :

1) Fuzzifikasi
   Proses fuzzifikasi adalah proses mengubah variabel non *fuzzy* (numerik) menjadi variabel *fuzzy* (variabel linguistik). Sistem inferensi bekerja dengan aturan dan *input fuzzy*. Memodifikasi *input crisp* menggunakan fungsi keanggotaan sehingga dapat digunakan oleh *rule base.*
2) *Rule Base*
   Terdiri dari dasar aturan dari seperangkat aturan seperti IF – THEN. Unit basis pengetahuan dalam logika *fuzzy* terdiri dari dua bagian sebagai berikut.
   a. Basis data (*data base*) berisi fungsi – fungsi keanggotaan dari himpunan *fuzzy* yang terkait dengan variabel linguistik yang digunakan.

b. Basis aturan (*rule base*) berisi aturan-aturan berupa implikasi. FIS mengevaluasi aturan yang relevan sesuai dengan variabel *input* saat ini.

3) Defuzzifikasi
Defuzzifukasi digunakan untuk menghasilkan nilai variabel solusi yang diinginkan dari suatu daerah konesekuen *fuzzy*. Mengkonversi *output* dari mesin inferensi menjadi nilai numerik.

2.2.13.3 Fuzzy Inference System Mamdani

Metode *Mamdani* sering juga dikenal dengan nama metode Max-Min. Metode ini diperkenalkan oleh Ebrahim Mamdani pada tahun 1975 (Ross, 2010). Untuk mendapatkan *output* diperlukan beberapa tahapan, antara lain :

1. Pembentukan himpunan *fuzzy*.
Pada metode *Mamdani*, baik variabel *input* maupun variabel *output* dibagi menjadi satu atau lebih himpunan *fuzzy.*
2. Aplikasi Fungsi Implikasi
Pada metode *Mamdani*, fungsi implikasi yang digunakan adalah Min. Secara umum dapat dituliskan:

$$\mu_{A\cap B}[x] = \min(\mu_A[x], \mu_B[x]) \quad (2.1)$$

3. Komposisi Aturan

Tidak seperti penalaran monoton, apabila sistem terdiri dari beberapa aturan, maka inferensi diperoleh dari kumpulan dan kolerasi antar aturan. Ada beberapa metode yang digunakan dalam inferensi sistem *fuzzy*, yaitu:

a. Metode *Max* (*Maximum*)

Metode *Max* mengambil solusi himpunan *fuzzy* diperoleh dengan cara mengambil nilai maksimum aturan, kemudian menggunakannya untuk memodifikasi daerah *fuzzy*, dan mengaplikasikannya ke *output* dengan operator OR (union). Jika semua proposisi telah dievaluasi, maka *output* akan berisi suatu himpunan *fuzzy* yang merefleksikan kontribusi dari tiap-tiap proporsi. Secara umum dapat dituliskan:

$$\mu_{sf}[x_i] \leftarrow max(\mu_{sf}[x_i], \mu_{kf}[x_i]) \quad (2.2)$$

dengan :

$\mu_{sf}[x_i]$ = nilai keanggotaan solusi *fuzzy* sampai aturan ke-i

$\mu_{kf}[x_i]$ = nilai keanggotaan konsekuensi *fuzzy* sampai aturan ke-i

b. Metode *Additive* (*Sum*)

Metode *Additive* mengambil solusi himpunan *fuzzy* diperoleh dengan cara melakukan *bounded-sum* terhadap semua *output* daerah *fuzzy* (Higuera, et al., 2013). Secara umum dituliskan sebagai berikut :

$$\mu_{sf}[x_i] \leftarrow min\left(1, \mu_{sf}[x_i] + \mu_{kf}[x_i]\right) \quad (2.3)$$

dengan :

$\mu_{sf}[x_i]$ = nilai keanggotaan solusi *fuzzy* sampai aturan ke-i

$\mu_{kf}[x_i]$ = nilai keanggotaan konsekuensi *fuzzy* sampai aturan ke-i

4. Penegasan (de*fuzzy*fikasi)

Input dari proses de*fuzzy*fikasi adalah suatu himpunan *fuzzy* yang diperoleh dari komposisi aturan-aturan *fuzzy*, sedangkan *output* yang dihasilkan merupakan suatu bilangan pada domain himpunan *fuzzy* dalam range tertentu sebagai *output* (Higuera, et al., 2013). Ada beberapa metode defuzzifikasi pada komposisi aturan *Mamdani*, antara lain :

a. Metode *Centroid* (*Composite Moment*)

Pada metode *Centroid* solusi *crisp* diperoleh dengan cara mengambil titik pusat daerah *fuzzy*. Secara umum dapat dituliskan:

$$z^* = \frac{\int z \cdot \mu_C(z) dz}{\int \mu_C(z)} \quad (2.4)$$

$$z^* = \frac{\sum_{j=1}^{n} z_j \cdot \mu_C(z_j)}{\sum_{j=1}^{n} \mu_C(z_j)} \quad (2.5)$$

b. Metode Bisektor

Pada metode bisektor solusi *crisp* diperoleh dengan cara mengambil nilai pada domain yang memiliki nilai keanggotaan setengah dari jumlah total nilai keanggotaan pada daerah *fuzzy*.

c. Metode *Mean of Maximum* (MOM)

Pada metode MOM solusi *crisp* diperoleh dengan cara mengambil nilai rata-rata domain yang memiliki nilai keanggotaan maksimum.

d. Metode *Largest of Maximum* (LOM)

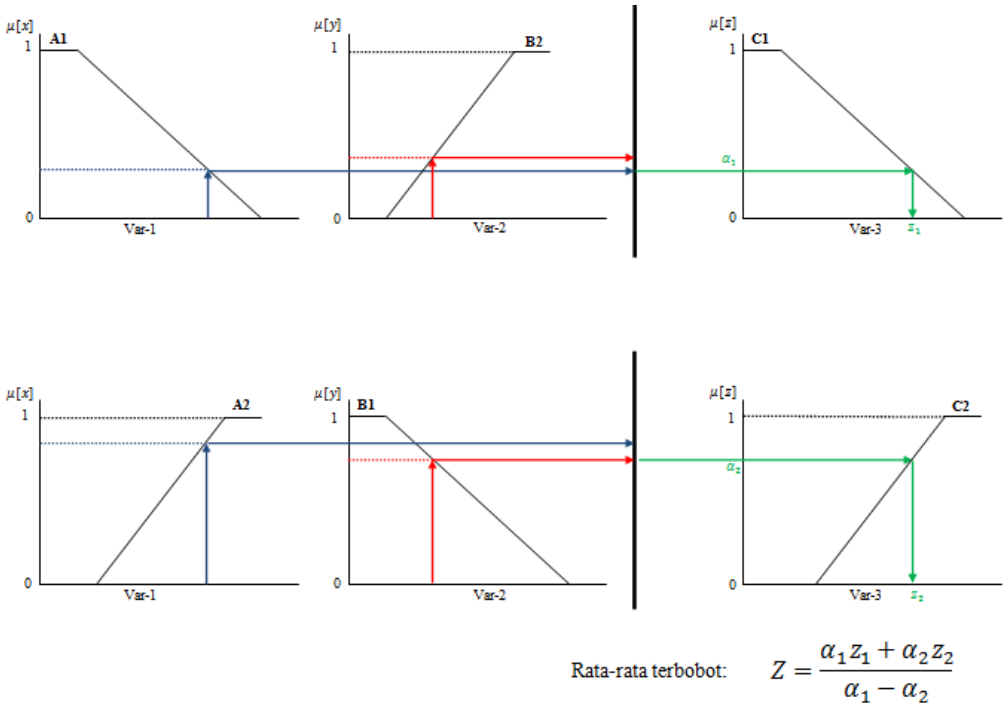
Pada metode LOM solusi *crisp* diperoleh dengan cara mengambil nilai terbesar dari domain yang memiliki nilai keanggotaan maksimum.

e. Metode *Smallest of Maximum* (SOM)

Pada metode SOM solusi *crisp* diperoleh dengan cara mengambil nilai terkeceil dari domain yang memiliki nilai keanggotaan maksimum.

2.2.13.4 Fuzzy Inference System Tsukamoto

Pada Metode *Tsukamoto*, setiap konsekuen pada aturan yang berbentuk *IF-THEN* harus direpresentasikan dengan suatu himpunan *fuzzy* dengan keanggotaan yang monoton. Sebagai hasilnya, *output* hasil inferensi dari tiap-tiap aturan diberikan secara tegas (*crisp*) berdasarkan α-predikat (*fire strength*) (Kusumadewi, 2003). Hasil akhirnya diperoleh dengan menggunakan rata-rata terbobot seperti gambar berikut.



Gambar 2.4 Inferensi Metode Tsukamoto

Sumber : (Kusumadewi, 2003)

Pada gambar di atas, nilai keanggotaan anteseden dari operasi *AND* dari aturan *fuzzy* [R1] adalah nilai paling kecil antara nilai keanggotaan A1 dari Var-1 dengan nilai keanggotaan B2 dari Var-2 dan antaseden dari aturan *fuzzy* [R2] adalah nilai paling kecil antara nilai keanggotaan A2 dari Var-1 dengan nilai keanggotaan B1 dari Var-2. Selanjutnya, nilai keanggotaan A2 dari Var-1 dengan nilai keanggotaan B1 dari Var-2. Selanjutnya, nilai keanggotaan antaseden dari aturan *fuzzy* [R1] dan [R2] masing-masing disebut dengan $\alpha 1$ dan $\alpha 2$ kemudian disubstitusikan pada fungsi keanggotaan himpunan C1 dan C2 sesuai aturan *fuzzy* [R1] dan [R2] untuk memperoleh nilai z1 dan z2, yaitu nilai z (nilai perkiraan produksi) untuk aturan *fuzzy* [R1] dan [R2]. Untuk memperoleh nilai *output* crisp/nilai tegas Z, dicari dengan cara mengubah *input* (berupa himpunan *fuzzy* yang diperoleh dari komposisi aturan-aturan *fuzzy*) menjadi suatu bilangan pada domain himpunan *fuzzy* tersebut. Cara ini disebut dengan metode defuzzifikasi (penegasan). Metode defuzzifikasi yang digunakan dalam metode Tsukamoto adalah metode defuzzifikasi rata-rata terpusat.