BAB 2

MEMAHAMI *LAYER* 2 TCP/IP *DATALINK LAYER*, PROTOKOL ICMP, DASAR LOGIKA *FUZZY* DAN *TRAFFIC SHAPER*

2.1 *Layer* 2 TCP/IP, *Datalink Layer*

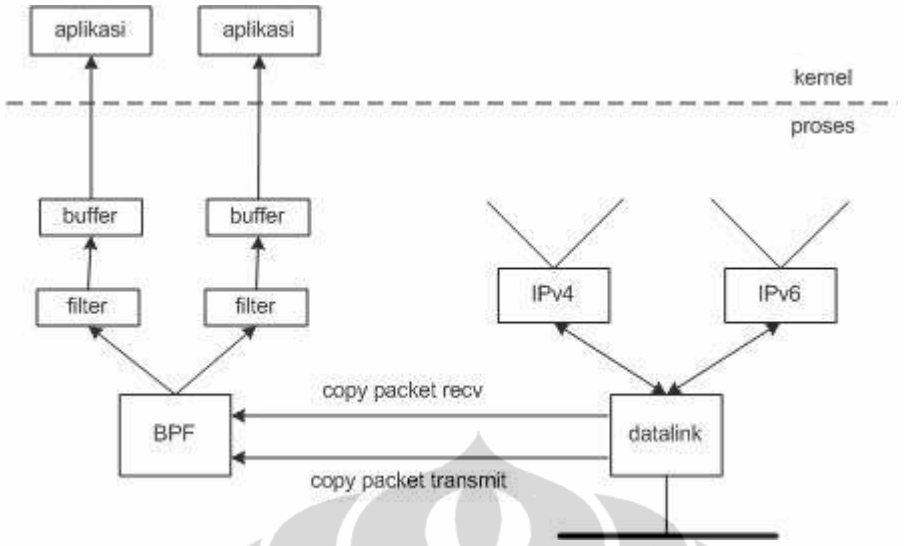
Akses *capture* data (*read/write*) ke *layer* 2 TCP/IP (*datalink layer*) telah disediakan oleh semua sistem operasi sekarang ini, baik yang berbasis *Microkernel* (proses dikerjakan sendiri-sendiri) maupun *Monolithic system* (keseluruhan sistem terdiri dari satu *single* a.out). Keduanya telah sama-sama menyediakan fitur ini. Beberapa *tool* populer yang mampu mengamati *packet* yang diterima secara *realtime* oleh *datalink* adalah tcpdump, iftop, trafshow dan iptraf (linux).

Metode umum yang digunakan untuk melakukan akses ke *datalink layer* antara lain: *BSD Packet Filter* (BPF) pada Unix, *Datalink Provider Interface* (DLPI) pada HP-UX dan Solaris, NIT pada SunOS, dan SOCK_PACKET, PF_PACKET pada Linux. *Library independent* yang dapat digunakan secara bebas yaitu pcap (*packet capture library*).

Terkait dengan penulisan laporan ini, metode akses yang digunakan adalah BPF, karena sistem operasi yang digunakan berbasis Unix FreeBSD, dan memanfaatkan *library packet capture (libpcap)* yang telah disediakan dalam modul perl yang dapat di *download* melalui *Comprehensive Perl Archive Network* (CPAN: http://www.cpan.org) dalam bentuk Pcap, RawIP, PcapUtils dan NetPacket (*Copyright* (c) 1995,1996,1997,1998,1999 ANU *and* CSIRO).

2.1.1 *BSD Packet Filter* (BPF)

Setiap *driver datalink* memanggil BPF tepat sebelum *packet* ditransmisikan dan tepat sesudah *packet* diterima.



Gambar 2.1 *Packet capture* menggunakan BPF [12]

Gambar 2.1 menjelaskan *packet* data masuk melalui *datalink device* /dev/bpf (diaktifkan dengan fungsi *lookupdev* dan *lookupnet (net* dan *mask)).* Kemudian fungsi *open_live* untuk *sniffing*. Dari BPF perlu mengaktifkan fungsi *compile* dan *setfilter* (TCP/UDP/ICMP) yang kemudian di-*queue* didalam *buffer* (dalam fungsi *loop* (*callback*)). Selanjutnya dibaca oleh aplikasi. Terakhir fungsi *close* untuk menutup *device*.

Alasan mengapa memanggil BPF secepat mungkin setelah penerimaan dan selama mungkin sebelum transmisi adalah untuk memperoleh akurasi *timestamp* (waktu ketukan/tanda). Kelebihan dari BPF adalah kapabilitasnya dalam *filtering*. Setiap aplikasi yang membuka sebuah *device* BPF dapat me-*load filter* sendiri.
Berikut tiga teknik yang digunakan oleh BPF untuk mengurangi *overhead*:

- BPF melakukan *filtering* didalam *kernel*, dengan maksud untuk meminimalkan jumlah data yang di*copy* dari BPF ke aplikasi. Proses meng*copy* dari *kernel space* ke *user space* sangatlah membebani. Jika setiap paket di*copy*, BPF akan mengalami masalah dengan *fast datalink*.
- Hanya per porsi dari setiap *packet* yang dilewatkan oleh BPF ke aplikasi. Hal ini disebut dengan *shapshot length* atau *snaplen*. Kebanyakan aplikasi hanya mengambil *packet header*, bukan *packet* data. Hal ini juga mengurangi keseluruhan data yang di*copy* oleh BPF ke aplikasi. Contohnya tcpdump, *default*nya 96, terdiri dari 14 byte *ethernet header*,

40 byte IPv6 *header*, 20 byte TCP *header* dan 22 byte data. Jika ingin memperoleh tambahan informasi untuk protokol lain (contoh: DNS dan NFS), maka memerlukan penambahan (*increase*) harga ini ketika tcpdump dijalankan.

- *Buffer* BPF di*copy*kan ke aplikasi hanya ketika *buffer* sudah penuh atau ketika *read timeout* sudah *expire*. Harga *timeout* ini dapat di-*seting* melalui aplikasi. Seperti contohnya tcpdump melakukan *set timeout* 1000 ms. Tujuan dari *buffering* adalah untuk mengurangi jumlah *sistem call*. *Sistem call* mempunyai *overhead*, jika *sistem call* berkurang maka akan mengurangi *overhead*.

Untuk mengakses BPF, harus membuka (*open*) *device* BPF, yang secara *default* tidak terbuka, contohnya /dev/bpf0, jika *return error* EBUSY, maka dapat menggunakan /dev/bpf1, dan seterusnya. Sekali *device* terbuka, maka puluhan perintah *ioctl* melakukan *set* karakteristik *device*, me-*load filter*, *set read timeout*, *set buffer size*, *attach datalink* ke *device* BPF, meng-*enable promiscuous mode*, dan seterusnya. Selanjutnya *I/O* akan melakukan proses *read* dan *write*.

2.1.2 Libpcap

Packet capture library (*libpcap*) melayani implementasi akses yang *independent* ke fasilitas *packet capture* yang disediakan oleh OS. Sekarang ini hanya menyediakan fasilitas membaca (*read*) saja meskipun sudah ada beberapa sistem yang menyediakan *write datalink*. *Library* ini digunakan oleh tcpdump, *man* PCAP(3).

Publish secara lengkap mengenai *library* libpcap:

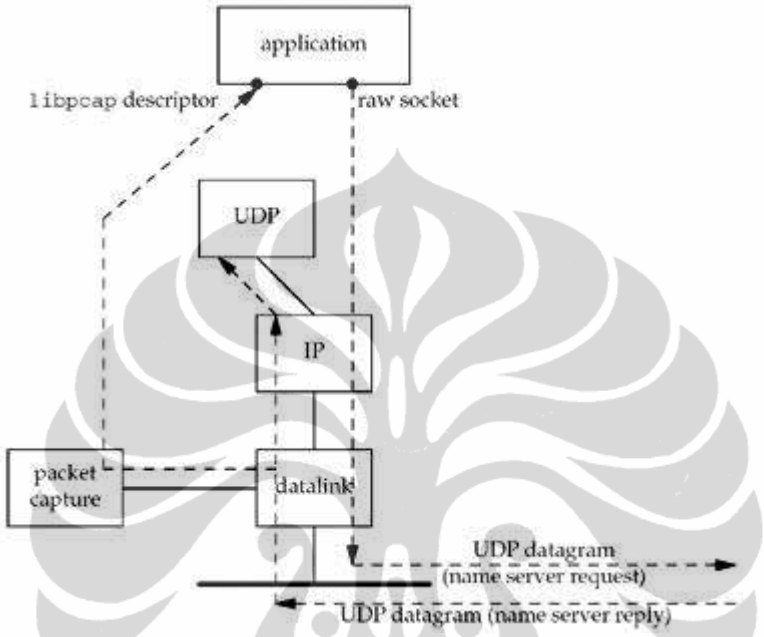
- libpcap: http://www.tcpdump.org/

2.1.3 *Checksum*

Banyak sistem sekarang ini menerapkan *checksum*. Adapun kegunaan dari *checksum* ini adalah untuk memastikan data *corrupt* atau tidak, atau data sesuai dengan *format*nya atau tidak, atau mengalami modifikasi ditengah jalan. Aplikasi FTS menerapkan *checksum* ICMP, karena yang digunakan untuk testing koneksi adalah ICMP. Sebab lain mengapa memerlukan *checksum* adalah jika data yang

tidak melalui *checksum* mengalami *corrupt*, maka *database server* juga akan *corrupt*.

Berikut adalah contoh diagram *checksum* dengan menggunakan UDP *checksum* dalam melakukan *request nameserver*:



Gambar 2.2 UDP *checksum* atas *request nameserver* [12]

Data yang akan dikirimkan akan dites dahulu dalam fungsi *checksum*, untuk menentukan apa *format*nya sudah benar dan data tidak *corrupt*, jika sudah benar maka akan dilakukan pengiriman ke jaringan.

2.2 Parameter Kinerja Jaringan

Ada banyak parameter untuk melihat kinerja jaringan. Disini hanya digunakan dua dari sekian banyak tersebut yaitu *round-trip time* dan *packet loss*.

Framework untuk parameter kinerja jaringan telah diatur di *IETF's IP Performance Metrics* (IPPM) *Working Group* di RFC 2330.

2.2.1 *Packet Loss*

Packet Loss merupakan probabilitas *packet* hilang sewaktu transit dari *source* ke *destination*. *A One-way Packet Loss Metric for IPPM* diatur di RFC 2680. RFC 3357 berisi *One-way Loss Pattern Sample Metrics*.

Transfer bulk data memerlukan transmisi yang *reliable*. Pada situasi ini, jika banyak *packet* hilang, maka harus dilakukan *retransmit* sehingga mengurangi kinerja. Sebagai tambahan, protokol *sensitif congestion* seperti standar TCP menganggap bahwa *packet loss* mempengaruhi *congestion* yang direspon agar mengurangi *rate* transmisi.

Untuk aplikasi *realtime* seperti audio video, akibat dari *packet loss* dapat mengakibatkan penurunan *sound* atau kualitas *image*.

Di FTS, *packet loss* tersebut akan diolah menjadi *loss ratio*. Adapun formulanya:

$$LR = \left[\frac{NL}{NL + NR} \right]$$

LR = *Loss Ratio*

NL = Jumlah *packet loss*

NR = jumlah *packet* yang diterima

Karena dalam persentase, make nilai diatas dikalikan 100%.

Congestion dan *Error* adalah dua alasan utama *packet loss*

2.2.1.1 *Congestion*

Ketika *load packet* yang dikirimkan mengalami *"exceed*" dari kapasitas jaringan, *packet* akan diantrikan (*queue*) di *buffer*, *buffer* sendiri juga mempunyai kapasitas terbatas, *congestion* dapat menyebabkan *queue overflows*, yang mana dapat menyebabkan *packet drop*. *Congestion* dapat disebabkan oleh kondisi *overload* yang *moderate*/ sedang, kemudian pada *extended time* tiba-tiba datang jumlah *packet* yang besar dari *traffic burst*.

2.2.1.2 *Error*

Alasan lain dari *loss packet* adalah adanya "*corruption*", dimana beberapa bagian dari packet dimodifikasi sewaktu transit. *Corruption* yang terjadi pada sebuah *link*, umumnya dideteksi oleh *link-layer checksum* pada penerima akhir, yang mana *packet* akan di *discard*.

2.2.2 *Round-Trip Time* (RTT)

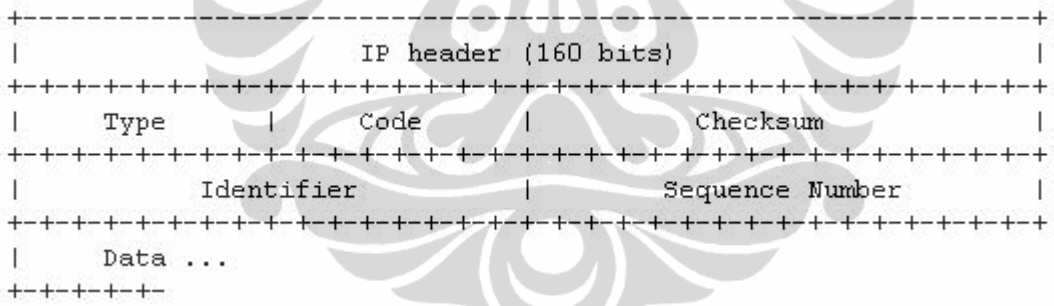
Round-trip time (RTT) adalah total waktu mulai *packet* dikirimkan sebuah *node* A sampai ke tujuan B kemudian merespon dan mengirim balik ke A. Dengan kata lain, *round-trip time* adalah "*sum one-way delay*" dari A ke B dan balik dari B ke A.

Harga RTT yang besar dapat menyebabkan masalah pada *Transmission Control Protocol* (TCP) dan protokol *transport* lain berbasis Windows. *Round-trip time* berpengaruh pada *throughput* yang diterima.

2.3 *Internet Control Message Protocol* (ICMP)

Sesuai dengan RFC 792, ICMP merupakan protokol yang didesain untuk mengirimkan *report error* (*feedback*), yang mana mengindikasikan apakah penerima (*host* atau *router*) "*reachable*" atau tidak setelah dikirimkan pesan ICMP oleh pengirim ke arah penerima.

Format ICMP *header*:



Gambar 2.3 *Format* ICMP *header* [9]

Tabel 2.1 *Format* ICMP *header* [10]

Bits	160-167	168-175	176-183	184-191
160	Type	Code	Checksum	
192	ID		Sequence	

ICMP *header* mulai dari *bit* 160 dari IP *Header*:

- *Type*: menunjukkan jenis tipe *message*. Berikut daftar tabel *Type*:

Tabel 2.2 *Type* ICMP *header* [10]

Type	Deskripsi
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect
8	Echo atau Echo Request
9	*Router* Advertisement
10	*Router* Solicitation
11	Time Exceeded
12	Parameter Problem: Bad IP header
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply

- *Code*: menunjukkan informasi tambahan dari *Type*. Di daftar tabel berikut tidak diulas semuanya mengenai *code*. Karena aplikasi FTS tidak menampilkan *error code* (yang berhubungan dengan *Type* 3 dengan tujuan mempercepat proses aplikasi dan menghindari kegagalan program).

  Berikut *Code* yang berhubungan dengan *Type* 3 *(Destination Unreachable)*:

Tabel 2.3 *Code Type 3* ICMP *header* [20]

Type 3 (Destination Unreachable)	
Code	Deskripsi
0	Destination network unreachable
1	Destination host unreachable
2	Destination protocol unreachable
3	Destination port unreachable
4	Fragmentation required, and DF flags set

Type 3 (Destination Unreachable)	
Code	Deskripsi
5	Source route failed
6	Destination network unknown
7	Destination host unknown
8	Source host isolated
9	Network administratively prohibited
10	Host administratively prohibited
11	Network unreachable for TOS
12	Host unreachable for TOS
13	Communication administratively prohibited

- *Checksum*: berfungsi untuk *error checking*, apakah data *corrupt* atau tidak yang merupakan penjumlahan dari ICMP *header* dan data. *Field* ini mempunyai harga 0.
- *Identifier (ID)*: berisi nilai *sequence* yang menunjukkan hubungan transaksi *Echo Request* dengan *Echo Reply*. Nilai *Identifier* antara *Echo Request* harus sama dengan *Echo Reply*.
- *Sequence Number*: berisi nilai yang fungsinya sama dengan *Identifier*.
- Data: data apa saja yang harus dikembalikan sama oleh *Echo Reply*. Jika data terlalu besar, maka IP *layer* akan melakukan fragmentasi pada paket data ini.

Padding Data :

- Linux dan Unix FreeBSD ”ping” memiliki blok *total size* 56 bytes dengan tambahan 8 *octet header*.
- Windows ”ping.exe” memiliki blok total *size* 32 bytes dengan tambahan 8 *octet header*.

2.4 Pengukuran *Latency/delay*

2.4.1 Ping

Ping adalah cara paling mudah dalam mengukur *delay*, yang memanfaatkan ICMP *echo request* (*type code 8*) dan ICMP *echo reply* (*type code*

0) kemudian menampilkan RTT (*Round Trip Time*) antara mesin *host* dan target. Ping dapat mengukur *packet loss* dengan mengirimkan satu *set packet* dari sebuah *host* ke target kemudian diambil perbandingan antara jumlah *packet* yang diterima dengan jumlah *packet* yang terkirim.

2.4.2 Traceroute

Program traceroute ditulis pertama kali oleh Van Jacobson (1988), *man* TRACEROUTE(8). Program ini mengirimkan ”*probe*” *packet* dengan memakai dasar harga TTL (*Time To Live*), naik (*increment*) 1, menggunakan pesan ICMP ”*Time Exceeded*” untuk mendeteksi jumlah *hop* ke arah target, dimana juga melakukan ”*records time*” tiap-tiap *hop,* kemudian menampilkan *loss* dan tipe kegagalan lain sesuai dengan jalur yang dilewati. Hal ini penting sebagai catatan untuk mengetahui node sepanjang path.

Traceroute banyak digunakan untuk mengetahui jalur track packet melalui *Internet* sampai ke target. *Packet* UDP dikirimkan sebagai ”*probe*” (biasanya pada *port* 33434-33525) dengan *field* TTL (*Time-To-Live*) di IP *header* terus bertambah (*increase*) 1 sampai ke *host* terakhir dicapai. Host menerima ”*LISTENS*” ICMP *Time Exceeded* sebagai bentuk respon dari tiap-tiap *router/end host*. Untuk mengetahui apakah *packet* benar-benar diterima oleh *end host*, seharusnya *end host* dapat melakukan ”*LISTENS*” pada *port range* tersebut. *Output* dari program traceroute menampilkan tiap-tiap *host* yang dilewati packet, dan juga menampilkan RTT tiap *gateway end-route* (implementasi di *NIX, traceroute mengirimkan UDP *probe packet* sedangkan di MS Windows traceroute mengirimkan ICMP *echo probes*, meskipun begitu di *NIX juga dapat mengirimkan pesan ICMP *echo* dengan menggunakan *flag* ”-I” atau ”-P icmp”. Seperti layaknya ping, sekarang ini ICMP dan UDP banyak di *block* oleh *firewall*.

Di laporan ini, untuk mengatasi *blocking* ini (jika *end host* menerapkan *firewall*/ atau menggunakan *antivirus firewall* terhadap ICMP), maka FTS akan berusaha mengambil informasi (berdasarkan *track*) *router* terakhir yang mampu di “*probe*”, dihitung dari mesin yang menggunakan aplikasi FTS (dengan anggapan ICMP *Time Exceeded* dengan *type code* ICMP 11 di izinkan).

2.5 IPFW dan Dummynet

FreeBSD mempunyai 3 *packet filter* antara lain: IPFW (BSD *origin*), PF (OpenBSD) dan IPFILTER (Darren Reed).

man IPFW(8), PFCTL(8), IPF(8)

FTS menggunakan IPFW sebagai *traffic shaper*. IPFW *utility* merupakan *user interface* untuk pengontrolan, *man* IPFW(4) *firewall* dan *man* dummynet(4) *traffic shaper*, *bandwidth manager* dan *delay emulator*. Konfigurasi IPFW atau *ruleset* dibuat berdasarkan *list rule* nomor 1 s/d 65535. *Packet* yang melalui IPFW berurutan di protokol *stack* berdasarkan nomor *rule*. *Default rule* 65535 tidak dapat dimodifikasi (tergantung seting kompilasi *kernel*nya dalam arti *allow* atau *deny*). FTS menerapkan *allow*, dengan kompilasi *kernel*:

options IPFIREWALL_DEFAULT_TO_ACCEPT

FreeBSD dummynet mempunyai fitur antara lain:

- Terintegrasi dengan IPFW untuk klasifikasi *packet*.
- Dapat digunakan pada *ingress*/ *egress*.
- Abstraksi/ fitur:
    - *Pipe*
        - Memberikan *fixed channel bandwidth*.
        - Memberikan nilai variabel pada *queue size*, *delay*, *random packet loss*.
    - *Queue*
        - *Queue packet*.
        - *Weighted* (*weight* bukanlah *priority*).
        - *Share bandwidth pipe*, yang mana berhubungan dengan *weigth*.
- WF2Q+ digunakan untuk *queueing discipline*.

2.6 Logika *Fuzzy*

Logika *fuzzy* adalah suatu cara memetakan suatu ruang *input* ke dalam suatu ruang *output*. Ditengah antara ruang *input* dan *output* tersebut, terdapat

kotak hitam yang merupakan proses logika dalam membentuk kesimpulan atau inferensi.

Alasan mengapa penulis menggunakan logika *fuzzy* adalah karena konsep ini sangat fleksibel berdasarkan pada bahasa alami (*IF-THEN*) dan mampu membangun kesimpulan tanpa melalui proses pelatihan seperti jaringan syaraf tiruan. Sudah banyak penerapan logika *fuzzy* dalam dunia nyata seperti contohnya mesin cuci Matsushita, kereta bawah tanah Sendai untuk otomatis pemberhentian, dan pada mobil Nissan untuk transmisi otomatisnya.

2.6.1 Himpunan *Fuzzy*

Himpunan tegas (*crisp*) mempunyai nilai keanggotaan nol (0) dan satu (1). Nilai keanggotaan juga bisa disebut derajat keanggotaan dengan symbol μ yang merepresentasikan nilai pada ordinat (y) dalam koordinat *cartesian*. Nilai nol (0) menunjukkan bahwa suatu item sudah diluar anggota himpunan, dan nilai satu (1) menunjukkan item menjadi anggota himpunan.

Jadi, dalam *fuzzy* ada 2 istilah pengelompokan himpunan, yaitu himpunan tegas (*crisp*) yang mempunyai nilai 0 dan 1, dan himpunan *fuzzy* yang mempunyai nilai keanggotaan antara 0 dan 1 (nilai pecahan).

Istilah lain adalah variabel *fuzzy* yang merupakan variabel yang akan dibahas dalam sistem *fuzzy*, contohnya: *round-trip time*, jumlah *hop* dan *loss ratio*.

2.6.2 Fungsi Keanggotaan

Fungsi keanggotaan (*membership function*) adalah pemetaan titik-titik input kedalam nilai keanggotaan (derajat keanggotaan) dalam rentang 0 s/d 1 dalam suatu kurva. Salah satu cara untuk memperoleh nilai keanggotaan adalah dengan menggunakan pendekatan fungsi. Ada beberapa fungsi yang digunakan :

a. Representasi Linear
b. Representasi Kurva Segitiga
c. Representasi Kurva Trapesium
d. Representasi Kurva Bentuk Bahu
e. Representasi Kurva-S

f. Representasi Kurva Bentuk Lonceng (*Bell Curve*)

g. Koordinat Keanggotaan

FTS menggunakan (a), dengan alasan paling mudah dan cepat perhitungan (aplikasi membutuhkan *realtime*). Meskipun tidak sampai presisi seperti Representasi Kurva Bentuk Lonceng (f), akan tetapi nilai harga pecahan untuk bandwidth tidaklah berpengaruh banyak (*Mean Opinion Scores (MOS)*). Ambil contohnya: alokasi bandwidth 64kbps tidaklah jauh beda dengan 63kbps karena bandwidth mempunyai sifat karakteristik yang *fluktuatif*.

2.6.3 Fungsi Implikasi

Tiap-tiap *rule* (aturan/proposisi) pada *fuzzy* akan berhubungan dengan suatu relasi. Bentuk umum dari aturan dalam fungsi implikasi adalah:

IF x is A *THEN* y is B

x dan y merupakan skalar, sedangkan A dan B himpunan *fuzzy*. Proposisi yang mengikuti *IF* disebut *anteseden*, sedangkan yang mengikuti *THEN* disebut *konsekuen*.

Secara umum 2 fungsi implikasi yang dapat digunakan, yaitu:

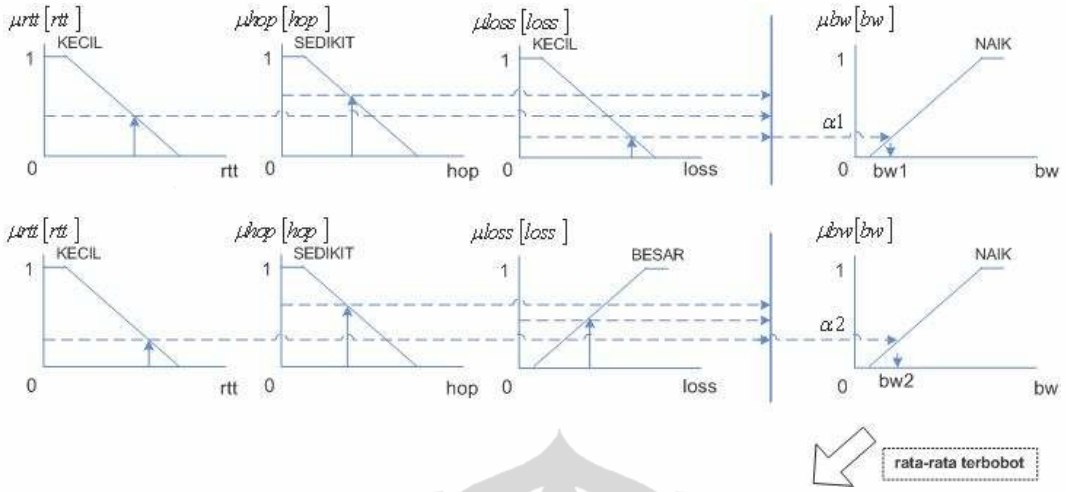
a. *Min (minimum)*. Fungsi ini akan memotong output himpunan *fuzzy*.

b. *Dot (product)*. Fungsi ini akan menskala output himpunan *fuzzy*.

Berikut contoh pengambilan inferensi berdasarkan metode *Tsukamoto* yang diterapkan pada 2 *rule* di FTS, (FTS mempunyai 8 buah *rule*).

Jika **rtt KECIL** dan **hop SEDIKIT** dan **loss KECIL**, maka **bw NAIK**
Jika **rtt KECIL** dan **hop SEDIKIT** dan **loss BESAR**, maka **bw NAIK**

(Istilah KECIL, SEDIKIT, TURUN merepresentasikan grafik *cartesian* dengan ordinat menurun sepanjang absis positif, dan istilah BESAR, BANYAK, NAIK merepresentasikan ordinat naik sepanjang absis positif).



Gambar 2.4 Inferensi FTS dengan Metode *Tsukamoto*

Rata-rata Terbobot (pada gambar diatas), [Persamaan 2.1 rata-rata terbobot].

$$bw\_avg(rata-rata)terbobot = \frac{\alpha 1 bw1 + \alpha 2 bw2}{\alpha 1 + \alpha 2}$$

Pada gambar tersebut nilai terkecil (min) dari *input*an (3 grafik sebelah kiri) akan memotong 1 grafik sebelah kanan. Nilai min tersebut menjadi sebuah nilai *output* yang disebut *Alpha* predikat dengan simbol α.

2.7 Manajemen Trafik

Manajemen trafik mempunyai tujuan kearah pelayanan *Quality of Services* (QoS) dari pengirim sampai penerima dengan mempertimbangkan efisiensi penggunaan *resource* jaringan. Manajemen trafik dapat diklasifikasikan menjadi tiga *level*: *packet level, flow level* dan *flow-aggregate level*.

Pada laporan tesis ini kami lebih fokus kearah *packet level* dengan penerapan di sisi *server*. Untuk mengatur bagaimana *packet* ditempatkan pada sistem *queue* disebut *queue management*. Beberapa cara tersebut dapat dilakukan dengan *First-In First-Out* (FIFO), *Fair Queue*, *Weighted Fair Queueing* dan *Random Early Detection*.

Manajemen trafik pada *flow level* menitikberatkan pada pengaturan *individual traffic flow* untuk memastikan QoS (contoh: *delay, jitter* dan *packet loss*) yang identik dengan adanya *admission control, policing* dan *traffic shaping* pada *open-loop control* diluar dari laporan tesis ini. Beberapa alasan antara lain:

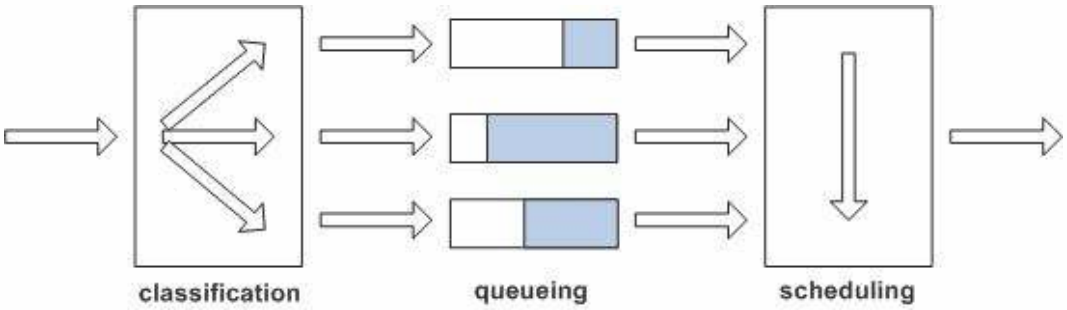
- Pada jaringan dengan adanya *admission control*, sebuah *source* perlu melakukan inisiasi sebuah *flow* baru yang mana memerlukan adanya izin (*permission*) dari *admission control* yang menentukan apakah *flow* diterima atau ditolak. *Admission control* sendiri harus mengetahui parameter trafik dan kebutuhan QoS setiap *flow*. Typical parameter trafik mengandung *peak rate* (bits/s atau bytes/s), *average rate* (bits/s atau bytes/s) dan maksimum *burst size* (bits, bytes atau *s*).
- *Device traffic shaping* di *flow level* perlu mengenali *delay packet* tertentu yang datang dan *schedule* meninggalkan dan juga keperluan *buffer* untuk *store packet*. Disini juga dikenal adanya garansi QoS dan *service scheduling*.
- FTS tidak memerlukan adanya perlakuan kelas-kelas layanan (*class of services*) dan *priority*, karena yang dilayani adalah satu jenis tipe aplikasi.

Adapun manajemen trafik pada *flow-aggregate level* seringkali disebut *traffic engineering*. Tujuan utama dari *traffic engineering* adalah *mapping aggregate flow* dalam jaringan sehingga *resource* dapat digunakan secara efisien. Disini dikenal dengan adanya *shortest-path*. Di laporan tesis ini tidak menggunakan *flow-aggregate level*.

2.8 *Traffic Control* dan *Queue Discipline*

Traffic control merupakan *set* mekanisme memperlakukan jenis trafik didalam jaringan, misalnya: menaikkan prioritas pada beberapa jenis trafik, membatasi *rate* trafik tertentu yang dikirim, melakukan *blocking* trafik tertentu (*packet filtering*).

- *Ingress* dan *Egress*
    - *Ingress* (trafik yang masuk *interface*)
        - *Classifying*, *dropping*.
    - *Egress* (trafik yang keluar *interface*)
        - *Queueing*.
        - Beberapa fungsi dapat diterapkan.



Gambar 2.5 *Traffic Control* [17]

- *Classification*
  - Mengamati konten dari *packet* dan menandainya kemudian menaruhnya pada kelas trafik.
- *Queueing*
  - *Packet* datang di antrikan menurut kelas-kelasnya.
- *Sceduling*
  - Mentransmit *packet* di*queue* berdasarkan *priority queue* dan melakukan *scheduling* yang berhubungan dengan *queue discipline*.

Berikut secara singkat mengenai *queueing disciplines*:

- *First In – First Out* (FIFO) atau *simple drop tail*
  - Memperlakukan *packet* berdasarkan waktu kedatangannya, *packet* akan di *discard* jika *buffer* penerima *full*.
- *Random Early Detection* (RED)
  - Teknik manajemen *buffer* yang melayani akses ke sistem FIFO yang akan mulai melakukan *dropping packet* sebelum *queue size* mencapai maksimum (*buffer overflows*). Dalam melakukan *drop*, sistem akan memberikan *feedback* informasi ke *source* agar *source* melakukan pengurangan transmisi *rate*nya.
- *Token Bucket Filter* (TBF)
  - Melakukan *shaper* secara *fixed rate*.
  - Melayani *burst* singkat diluar *fixed rate*.
- *Priority Scheduling*
  - Mengutamakan proses *packet* pada kelas *priority* lebih tinggi sebelum yang lebih rendah diproses.

- *Weighted Fair Queueing* (WFQ)
    - Setiap *user flow* mempunyai *logical buffer* sendiri dimana setiap *user flow* tersebut juga mempunyai *weight*, *weight* tersebut akan menentukan *share bandwidth*.
    - Menandai setiap *queue* dengan setiap *flow*.
    - *Weight* dapat ditentukan dengan setiap *queue*.