BAB 3
PROGRAM DAN *TESTBED*

Ada banyak teknik yang dilakukan untuk mengevaluasi kinerja suatu sistem. Teknik simulasi yang sering digunakan untuk menguji sistem karena termudah dan mampu memperoleh hasil dalam waktu yang singkat dengan biaya yang cukup rendah bukanlah teknik yang terbaik jika dibandingkan dengan testbed yang langsung melakukan pengujian pada kondisi sesungguhnya. Karena alasan inilah, maka akan dipilih *testbed* sebagai teknik untuk melakukan pengujian.

Tujuan dari penelitian ini adalah untuk mengatur kebutuhan trafik *bandwidth* disisi *streaming server* secara otomatis dengan menerapkan *quota bandwidth* tiap-tiap pengguna berdasarkan deteksi *datalink layer* (*sniff*, memperoleh *IP Address source, IP Address destination, port source* dan *port destination*), kemudian *server* akan mengevalusi tiap-tiap *IP Address* tersebut dengan mengirimkan tes kirim *packet* data dengan Ping dan Traceroute sehingga akan diperoleh *round-trip time*, jumlah *hop* dan *loss ratio* tiap-tiap pengguna, data informasi tersebut akan dimasukkan *database* MySQL (*backend*) dengan DBI sebagai *interface* yang kemudian langsung diolah dengan logika *fuzzy* sebelum menerapkan *traffic shaping* tiap pengguna.

Bagian pertama dari bab ini membahas rancangan konfigurasi *server* yang akan dilakukan berikut paket pendukungnya. Pada bagian selanjutnya dibahas mengenai model skenario *testbed* dan seting *router* (*routing* statik), diikuti dengan detail rancangan program dan *traffic shaping*.

3.1 Konfigurasi *Server*

Sistem Operasi yang akan digunakan adalah UNIX FreeBSD versi 6.2 dan 7.2. Adapun proses kompilasi *kernel* yang akan dilakukan sbb:

3.1.1 Kompilasi *Kernel*

Kompilasi standar tanpa menggunakan CVSUP (*Concurrent Versions System*), CVSUP merupakan *software source code version control system* yang

mana berfungsi untuk melakukan *tracking* perubahan yang terjadi pada *source code* dari pertama *source code* tersebut dibuat, *man* CVSUP(1).

Tabel 3.1 Kompilasi *Kernel*

options	IPFIREWALL
options	DUMMYNET
device	bpf

IPFW digunakan sebagai *ip filter & traffic shaper*, *man* IPFW(8), DUMMYNET(4) sebagai *traffic shaper*, *bandwidth manager* and *delay emulator*, sedangkan BPF berhubungan dengan *datalink layer*.

3.1.2 *Software Server* dan Modul CPAN

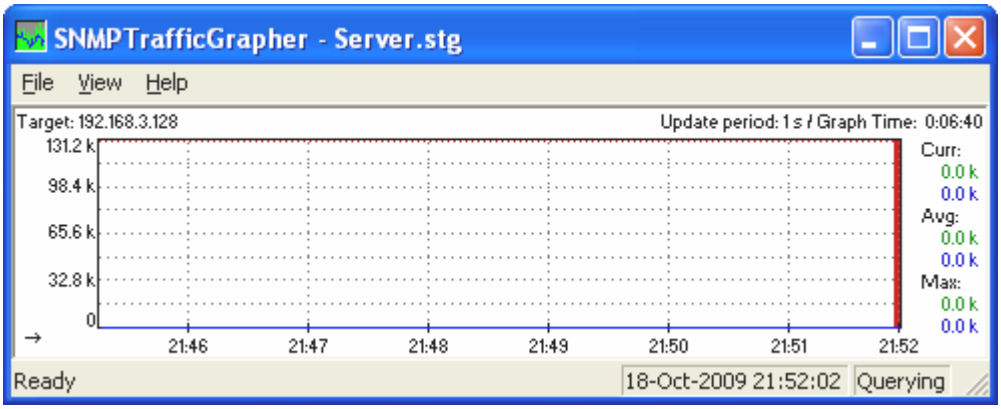
Beberapa paket *software server*, *codec*, *database* dan paket pendukung untuk *streaming server* akan diinstal langsung melalui *Internet* (*PORTS*), antara lain: apache, php, mysql, phpmotion, mencoder dan ffmpeg.

Paket pendukung FTS berupa modul CPAN. Paket modul CPAN antara lain: Net::RawIP, Net::Pcap, Net::PcapUtils, NetPacket, DBI, DBD::mysql, Net::SNMP, Net:::SNMP::Interfaces, Net::SNMP::HostInfo.

3.1.3 *Monitoring*

Monitoring yang akan dilakukan dengan menggunakan protokol SNMP. *Simple Network Management Protocol* (SNMP), pertama kali diperkenalkan pada 1988 sebagai standar dalam manajemen *IP device*, yang mengizinkan *device* di kontrol secara *remote*, *man* SNMPD(1).

Untuk *monitoring* berdasarkan *queue* IPFW, maka digunakan ipfwsnmp (Hideyuki Suzuki hideyuki@sat.t.u-tokyo.ac.jp) yang ditulis dalam bahasa perl. *Queue* setiap IPFW tersebut akan dijadikan *octet* terakhir dari OID yang akan di-*trap* oleh SNMP *Traffic Grapher* (STG) (berbasis Windows), sehingga diperoleh hasil *monitoring* yang *realtime*.



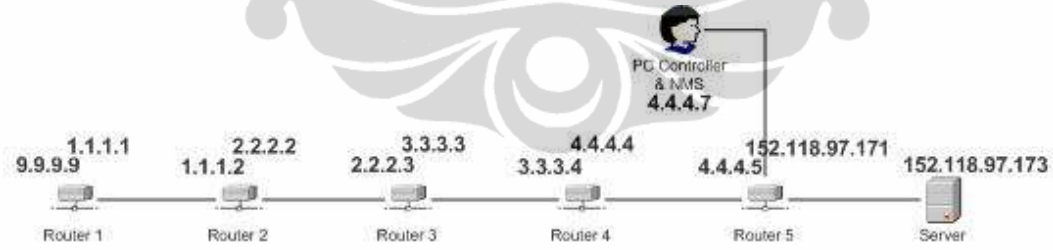
Gambar 3.1 *SNMP Traffic Grapher* (STG)

Penjelasan mengenai monitoring lebih detail pada akhir bab ini.

3.2 Model Skenario *Testbed*

Model skenario *testbed* terdiri dari 1 PC *Server*, 5 *router* dan 5 PC pengguna. Masing-masing *router* di*seting* dengan *routing* statik dan tidak menggunakan *Network Address Translation (NAT)*. NAT akan menimbulkan masalah dengan *trace IP Address pengguna*, karena *IP Address pengguna* akan bersifat *private*, sehingga yang berhasil di *probe* adalah *IP Address public router*.

Berikut *IP Address* dari *router*, *server* dan PC *controller* yang juga bertindak sebagai NMS:



Gambar 3.2 Topologi *Server*, *Router dan PC Controller* pada *testbed*

***Router*-1** terkoneksi ke *Router*-2

9.9.9.9 (*IP Address Router*-1 LAN *interface*)

1.1.1.1 (*IP Address Router*-1 WAN *interface*)

***Router*-2** terkoneksi ke *Router*-1 & *Router*-3

1.1.1.2 (*IP Address Router*-2 LAN *interface*)

2.2.2.2 (*IP Address Router*-2 WAN *interface*)

***Router*-3** terkoneksi ke *Router*-2 & *Router*-4

2.2.2.3 (*IP Address Router*-3 LAN *interface*)

3.3.3.3 (*IP Address Router*-3 WAN *interface*)

***Router*-4** terkoneksi ke *Router*-3 & *Router*-5

3.3.3.4 (*IP Address Router*-4 LAN *interface*)

4.4.4.4 (*IP Address Router*-4 WAN *interface*)

***Router*-5** terkoneksi ke *Router*-4

4.4.4.5 (*IP Address Router*-5 LAN *interface*)

152.118.97.171 (*IP Address Router*-5 WAN *interface*)

PC Controller & NMS terkoneksi ke *Router*-5

4.4.4.7 (*IP Address PC Controller & NMS*)

Server terkoneksi ke *Router*-5

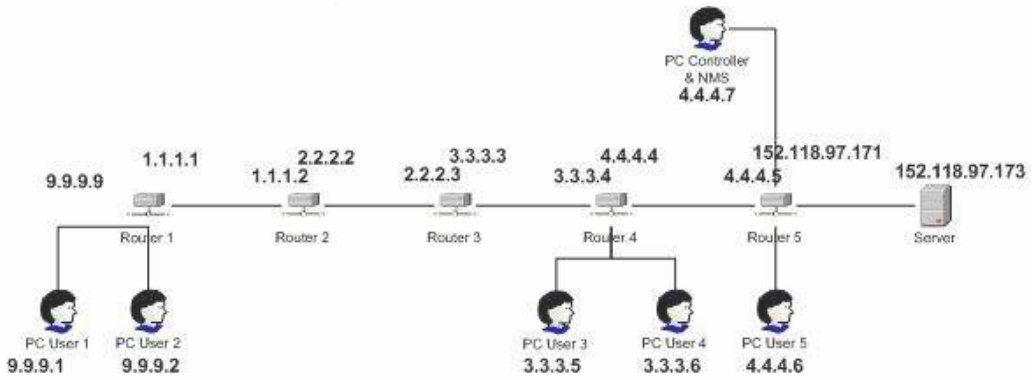
152.118.97.173 (*IP Address Server*)

Selain *routing default*, setiap *router* juga mempunyai *routing* yang diarahkan ke setiap *network* dalam *testbed*.

Seting IP Address tidak mengacu pada kelas-kelas *IP Address* (A,B,C), agar lebih mudah untuk dipahami dan di *trace* kesalahan, selain itu yang dibahas dalam laporan tesis ini bukanlah *seting* jaringan, akan tetapi *end-to-end*. Dengan anggapan disisi jaringan tidak mempunyai banyak gangguan.

3.2.1 Skenario 1

Skenario pertama ini bertujuan untuk mengamati *bandwidth* yang dialokasikan oleh *Server* untuk pengguna yang tidak terkonsentrasi/ acak.



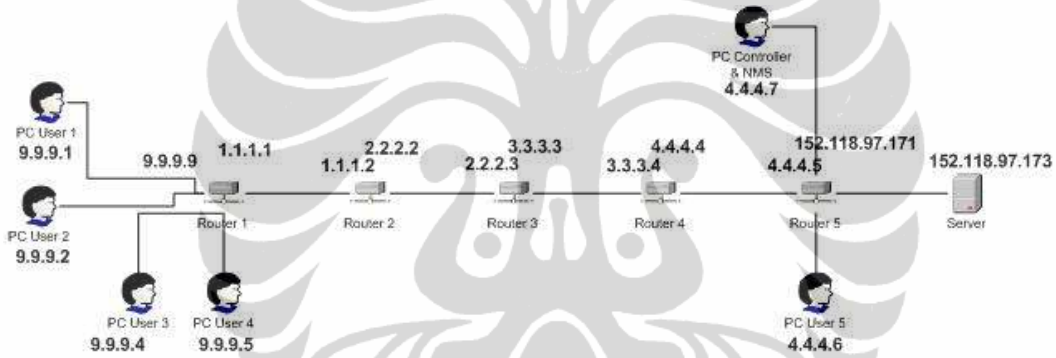
Gambar 3.2.1 Topologi *testbed* skenario 1

Dari topologi tersebut dapat dilihat bahwa ada 5 pengguna yang terkoneksi dimana letaknya tidak teratur. Pengguna 1 (9.9.9.1) dan Pengguna 2 (9.9.9.2) terkoneksi ke *Router-1*, Pengguna 3 (3.3.3.5) dan Pengguna 4 (3.3.3.6) terkoneksi ke *Router-4*. Pengguna 5 (4.4.4.6) terkoneksi ke *Router-5*.

Dari skenario 1 akan diamati tingkah laku dan alokasi *bandwidth* yang akan diperoleh setiap pengguna, pada waktu *Server* menggunakan program FTS, dan dengan pada waktu tidak menggunakan. Maksimum *bandwidth* yang dimiliki *Server* akan di-*seting* 768kbps (percobaan 1) dan 1024kbps (percobaan 2).

3.2.2 Skenario 2

Skenario kedua bertujuan untuk mengamati *bandwidth* yang dialokasikan oleh *Server* untuk pengguna yang terkonsentrasi.



Gambar 3.2.2 Topologi *testbed* skenario 2

4 Pengguna terkoneksi ke *Router-1*, sedangkan 1 Pengguna terkoneksi ke *Router-5*, (1 Pengguna tersebut ditujukan sebagai pembanding dengan koneksi paling bagus ke *Server*, yang seharusnya memperoleh alokasi bobot *bandwidth* terbesar).

Seperti pada skenario 1, skenario 2 ini juga akan mengamati tingkah laku dan alokasi *bandwidth* yang akan diperoleh setiap pengguna pada waktu *Server* menggunakan program FTS, dan dengan pada waktu tidak menggunakan. Maksimum *bandwidth* yang dimiliki *Server* akan di-seting 768kbps (percobaan 1) dan 1024kbps (percobaan 2).

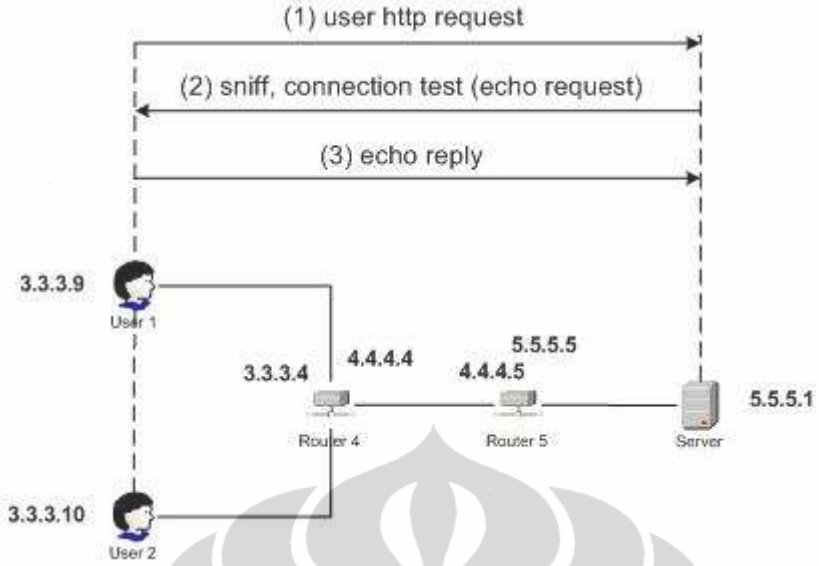
3.3 *Fuzzy Traffic shaper* (FTS)

FTS merupakan program yang dibuat pada laporan tesis ini sebagai aplikasi yang berfungsi untuk melakukan *traffic shaping* berdasarkan informasi yang diperoleh dari pengguna yang mengakses *streaming server*. Adapun informasi yang diperlukan *server* dari setiap pengguna berupa *round-trip time*, jumlah *hop* dan *loss ratio* dari trafik ICMP. Intinya jika koneksi bagus maka akan memperoleh *weight* lebih besar. Jumlah keseluruhan pengguna dalam waktu bersamaan juga dipertimbangkan sebagai pembanding akhir penentuan alokasi *quota bandwidth*. Sehingga diharapkan *server* dapat memberikan layanan yang optimal.

Program akan dibagi menjadi dua proses (sniff.pl dan hit.pl), yang pertama berfungsi sebagai *sniffing* terhadap *datalink layer* dan mengambil informasi *IP Address*, *mac address* berikut *port* dari pengguna (mendeteksi pengguna yang masuk ke *server* yang mana pengguna menggunakan protokol TCP), kemudian dari informasi tersebut, *server* akan melakukan "*probe*" informasi ke setiap pengguna (menggunakan protokol ICMP). Proses kedua berfungsi sebagai pengolah data dan *traffic shaper*.

Program dirancang dalam bahasa perl yang terdiri dari dua *file* eksekusi (sniff.pl dan hit.pl), dan 5 modul (configLIB.pm, fuzzyLIB.pm, icmpLIB.pm, ipfwLIB.pm dan traceLIB.pm). *File* lain dalam aplikasi yang berhubungan dengan *database* adalah sql_create_icmp.sql dan sql_delete_tabel_icmp.sql. Selain itu *file independent* juga disertakan sebagai fasilitas *debug* koneksi, baik koneksi yang berhubungan dengan jaringan maupun koneksi aplikasi dengan *database*.

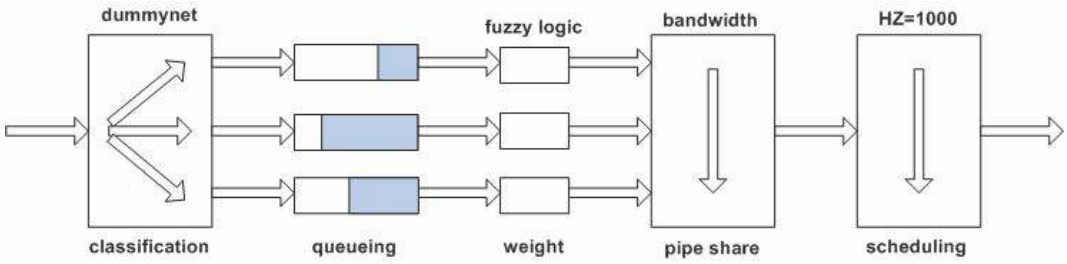
Pada sub bab ini akan dibahas secara detail mengenai cara memperoleh nilai variabel parameter kinerja jaringan dan pengolahan data menggunakan logika *fuzzy*, terakhir adalah *traffic shaper* dan *monitoring*.



Gambar 3.3 Proses *Routing* FTS

Detail proses:

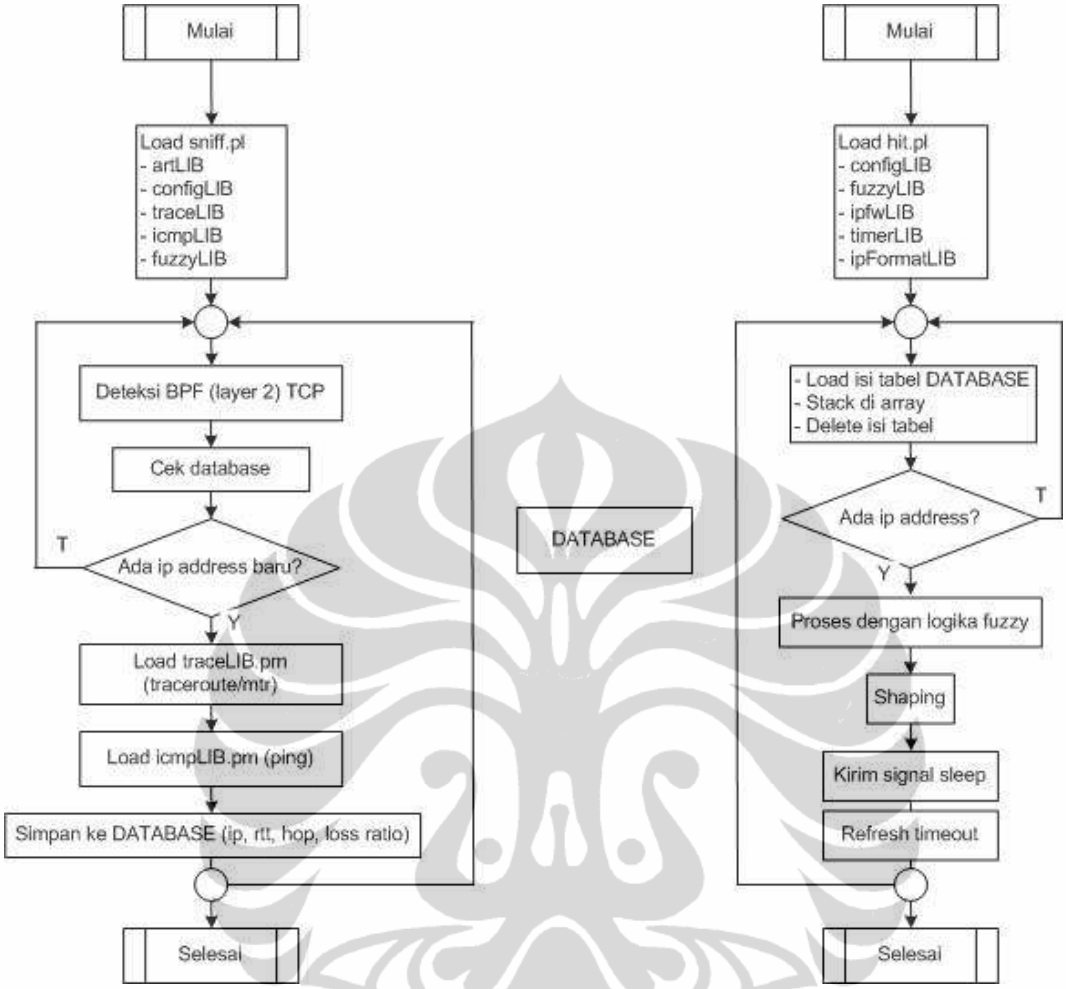
1) Pengguna melakukan *request* http melalui *browser*.
2) FTS *server* menangkap koneksi pengguna dengan melakukan *sniffing* untuk memperoleh *IP Address*, *port* dan *mac address* pengguna tersebut dan kemudian mengirimkan *icmp echo request* ke *IP Address* tersebut.
3) Pengguna mengirim *feedback* dengan mengirimkan *icmp echo reply* ke FTS *server*. Dalam kasus terdapat *antivirus / firewall* yang memblocking icmp disisi PC pengguna, maka *router* terakhir yang mampu di '*probe*" akan mengirimkan ICMP *time exceeded*.
4) FTS *server* menerima informasi dari pengguna berupa *round-trip time (rtt)*, jumlah *hop* dan *loss packet*. FTS akan mengolah *loss packet* menjadi *loss ratio* dan mengolah data-data tersebut dengan logika *fuzzy* untuk memperoleh nilai *weight* setiap *user flow* dan melakukan *traffic shaping* dengan menggunakan WF2Q+. Berikut *queue discipline* yang menjelaskan proses *traffic shaping* di FTS:



Gambar 3.4 FTS *Traffic Control*

- Dummynet menggunakan *firewall* (IPFW) untuk meng-*classify packet* dan membaginya menjadi *flow*.
- Sebuah *flow* dapat terdiri dari paket dari satu koneksi TCP, atau dari/ke *host/subnet* atau tipe protokol.
- Sebuah *queue* adalah sebuah abstraksi/fitur yang digunakan untuk implementasi WF2Q+ *policy*.
- *Queue* berhubungan dengan sebuah *weight* dan sebuah *pipe* referensi (*pipe* referensi bukanlah *pipe share*) untuk setiap *flow*.
- *Pipe* referensi tersebut kemudian mengacu ke *pipe share* untuk alokasi maksimum *bandwidth* yang dipunyai *server* FTS (*bandwidth* yang akan di*share*).
- Sebuah *weight* diperoleh dari proses perhitungan dengan logika *fuzzy*.
- *Delay* propagasi mengacu pada kompilasi *kernel* FreeBSD (HZ=1000) untuk mengurangi *granularity* menjadi 1ms atau lebih kecil. *Default* harga adalah 0 yang berarti tidak ada *delay*.

Berikut *Flowchart* FTS:



Gambar 3.5 Flowchart FTS

- sniff.pl
  - *Sniffing* paket data dengan memanfaatkan *library packet capture*. Adapun modul yang di*load* antara lain: Net::Pcap, Net::PcapUtils, NetPacket::Ethernet qw(:strip), NetPacket::IP qw(:strip), NetPacket::TCP dan NetPacket::Ethernet.
  - Melakukan *filter* TCP kemudian membaca setiap koneksi yang masuk ke *ethernet* (*datalink* atau *layer* 2 TCP/IP), proses akan dilakukan secara *looping*. Data sebisa mungkin masuk sebanyak-banyaknya, sehingga penulis tidak membatasi dengan variabel *timeout*. Informasi yang diperlukan oleh FTS adalah *IP Address source* (*user*), *Port source* (*user*), *IP Address destination* (*server*)

dan *Port destination* (*server*). Karena dalam penelitian ini *server* hanya melayani *port* 80 (HTTP) sebagai *web streaming server*, maka yang di baca adalah koneksi ke *port* 80 ke arah *server*.

- Sniff.pl akan menganalisa *database* dengan memanggil configLIB.pm, jika tidak terdapat pengguna dengan *IP Address* yang sama, maka sniff.pl akan me-*load file* traceLIB.pm dan icmpLIB.pm. Jika terdapat *IP Address* yang sama, maka sniff.pl akan *looping*. Dalam kasus jika tidak terdapat *IP Address* yang sama, maka sniff.pl akan memanggil traceLIB.pm.
- traceLIB.pm bertugas untuk melakukan *trace* berdasarkan *IP Address* dan memberi keluaran berupa *hop* dan *IP Address* terakhir yang mampu di ”*probe*”. Secara *default*, Unix menerapkan *Time-To-Live* (TTL) = 128, sedangkan Windows = 64. *Seting timeout* = 1 (1 detik).
- Setelah traceLIB.pm memberikan keluaran berupa *hop* dan *IP Address* terakhir yang berhasil di ”*probe*”, maka sniff.pl akan memanggil icmpLIB.pm (disini dibuat perulangan agar seorang *system administrator* pengguna aplikasi dapat leluasa menentukan berapa kali proses ping dilakukan sehingga bisa diambil keluaran *packet loss* sesuai dengan medan jaringannya). Penulis tidak menggunakan *tool* ping yang disediakan OS dengan maksud agar struktur ICMP *field* dapat dibuat sekecil-kecilnya selama sesuai dengan *checksum* (my $icmp_struct = "C2 n3 A";), sehingga susunan icmp struktur paket paling kecil yang di kirimkan ke target secepat mungkin dapat dikirim dan diterima dalam hitungan ms, selain itu panjang *packet* dalam ping *tool* yang disediakan OS *default* sebesar 56 (64 byte), yang merupakan penjumlahan 56+8 , sedangkan pada icmpLIB.pm, penulis memberikan harga panjang paket sebesar 2 (10 byte), 2+8 *octet header*. Berikut perbandingan antara *tool* ping OS dengan icmpLIB.pm:

Ping tool OS, jika panjang paket = 56 (**my $packetlen = 56;**)

FreeBSD# perl tes_icmp.pl

[1] [ip: 127.0.0.1] [rtt: 0.53 ms] [**bytes kirim: 64**] [bytes terima: 84]

[2] [ip: 127.0.0.1] [rtt: 0.47 ms] [**bytes kirim: 64**] [bytes terima: 84]

[3] [ip: 127.0.0.1] [rtt: 0.11 ms] [**bytes kirim: 64**] [bytes terima: 84]

Senilai dengan ping *tool* OS:

FreeBSD# ping 127.0.0.1

PING 127.0.0.1 (127.0.0.1): **56 data bytes**

64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.606 ms

64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.805 ms

64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.555 ms

icmpLIB.pm, jika panjang paket = 2 (**my $packetlen = 2;**)

FreeBSD# perl tes_icmp.pl

[1] [ip: 127.0.0.1] [rtt: 0.52 ms] [**bytes kirim: 10**] [bytes terima: 30]

[2] [ip: 127.0.0.1] [rtt: 0.92 ms] [**bytes kirim: 10**] [bytes terima: 30]

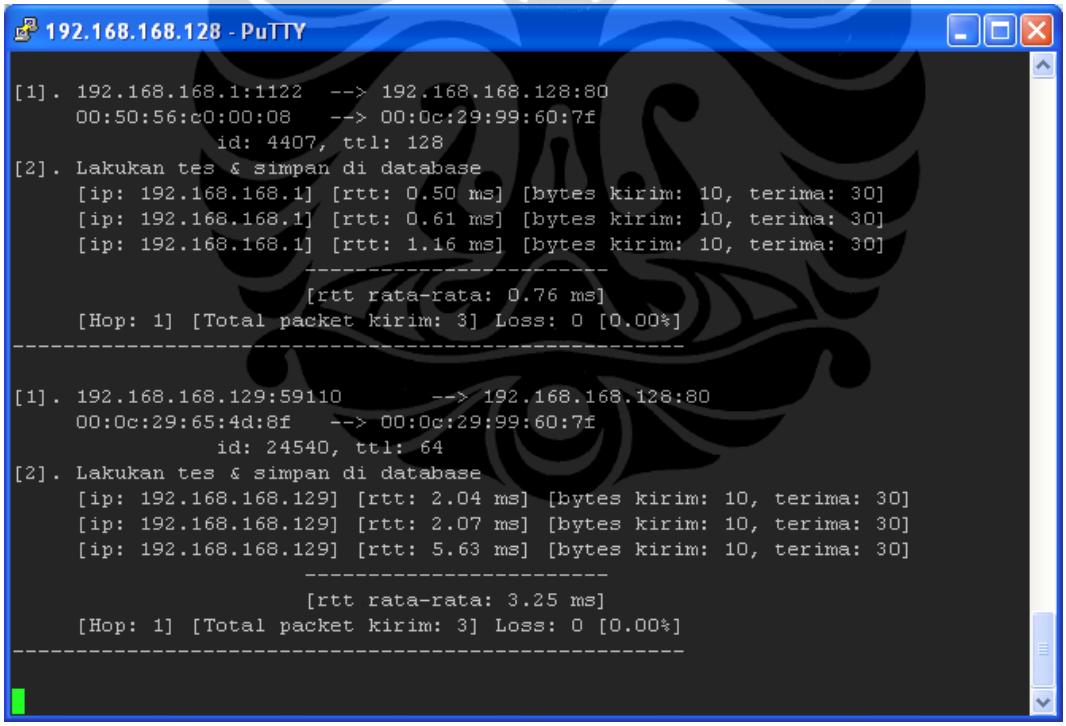
[3] [ip: 127.0.0.1] [rtt: 0.88 ms] [**bytes kirim: 10**] [bytes terima: 30]

Perbandingan besar paket yang dikirim tentunya akan berpengaruh jika koneksi padat, karena proses tes koneksi tentunya memakan *bandwidth* jaringan meskipun cepat.

- icmpLIB.pm bertugas untuk melakukan ping *packet* kearah *IP Address* yang dikeluarkan oleh traceLIB.pm. icmpLIB.pm menggunakan beberapa fungsi *socket*. SOCK_RAW dipergunakan karena data yang dikirim berupa ICMP, bukan TCP ataupun UDP. Beberapa keuntungan menggunakan SOCK_RAW yang tidak dimiliki SOCK_STREAM (TCP) dan SOCK_DGRAM (UDP) antara lain: *read/write* ICMPv4/v6 dan IGMPv4, *read/write* IPv4 *datagram* yang tidak diproses di *kernel* dan memperoleh (*Recall*) 8-bit IPv4 *field* protokol, terakhir proses dapat membuat sendiri IPv4 *header* menggunakan *options* IP_HDRINCL. Sedangkan PF_INET, inet_aton inet_ntoa sockaddr_in unpack_sockaddr_in

merupakan struktur *(struct)* dari IPv4 *Socket Address*. AF_INET dengan PF_INET adalah sama, di C menggunakan AF_INET sedang di Perl penulis menggunakan PF_INET. Dari icmpLIB.pm akan diperoleh *return* = 1, durasi waktu x 1000 = *rtt* dalam ms, *IP Address* pengguna, panjang paket kirim dan panjang paket terima. Hasil tersebut diserahkan ke sniff.pl.

- Dari proses traceLIB.pm diperoleh *hop*, dan dari icmpLIB diperoleh *rtt* dan *packet loss*, yang selanjutnya *packet loss* akan diolah dengan formula *loss ratio* di sniff.pl sehingga diperoleh tiga parameter kinerja jaringan yang siap untuk di simpan ke *database*.
- Proses diatas akan diulang terus begitu ada pengguna dengan *IP Address* baru (yang tidak ditemukan di *database*).



Gambar 3.6 Tampilan sniff.pl

- hit.pl
  - hit.pl merupakan *file* utama dari aplikasi ini, yang memanggil modul configLIB.pm, fuzzyLIB.pm dan ipfwLIB.pm. Pertama kali di *load* akan meminta *input* manual berupa *bandwidth minimum*

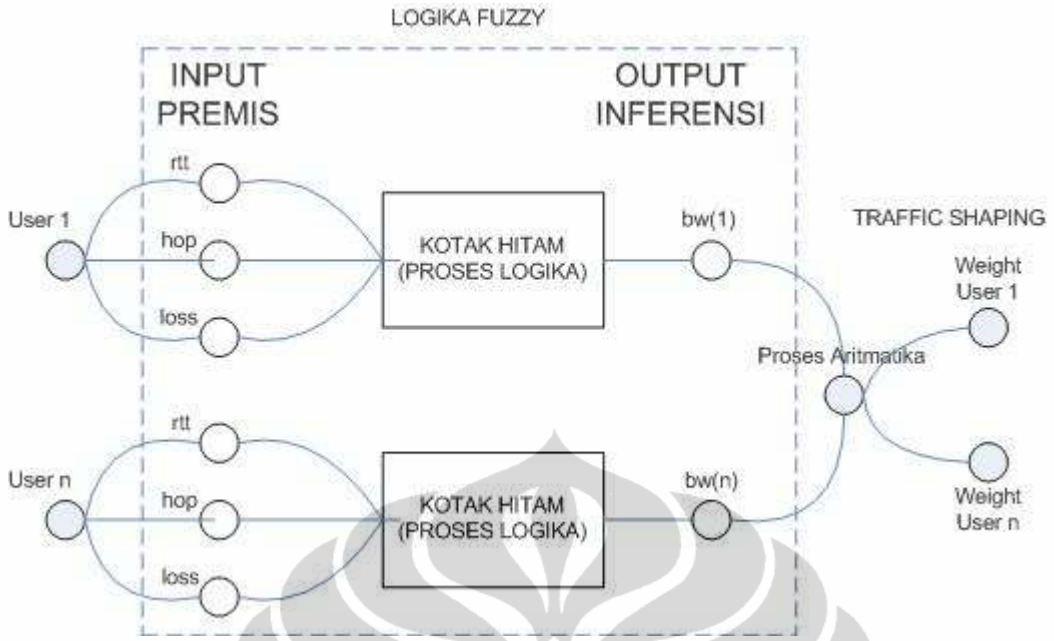
dan maksimum (keseluruhan) yang dialokasikan untuk *streaming server*. Seperti halnya sniff.pl, hit.pl diseting *looping*. *Subroutine* pertama yang di*load* adalah opendata(), yang mana berfungsi untuk memanggil *database* dan *searching* semua informasi yang dipunyai *database* (dengan fungsi() di configLIB.pm). Jika *database* mempunyai informasi (*IP Address*, *rtt*, *hop* dan *loss ratio*), maka akan disusun kedalam *stack array* (kumpulan informasi) @ip, @rtt, @hop dan @loss. Keluaran *return* berupa referensi \@ip, \@rtt. \@hop dan \@loss. Setelah semua informasi terambil, maka isi tabel *database* akan dihapus, untuk memberikan proses ke sniff.pl, sehingga *fresh IP Address* dapat di*input*kan kembali ke *database*.

- Selanjutnya dipanggil *subroutine* olahdata(). *Subroutine* ini menampilkan semua informasi yang berhasil diambil dari *database* secara *realtime* sebelum diproses menggunakan logika *fuzzy*.

3.3.1 Proses Logika *Fuzzy*

Inti dari laporan tesis ini terletak pada pengolahan data menggunakan metode logika *fuzzy*. Sebelum diproses, file modul fuzzyLIB.pm akan dipanggil. Di fuzzyLIB.pm selain perhitungan *fuzzy* juga dibuat fungsi metode *sorting* (*bubble sort*) yang berfungsi untuk menentukan nilai *minimum* ($min) dan *maximum* ($max). Setiap keluaran referensi *array* dari opendata() terlebih dahulu akan diolah dengan metode *sorting* untuk menentukan nilai terendah (*min*) dan nilai tertinggi (*max*) sebagai batas bawah dan batas atas dalam perhitungan.

Pada rancangan program ini dipergunakan *Worst-case Fair Weighted Fair Queueing* (WF2Q+) sebagai *queueing disciplines*. Berikut *rule* logika *fuzzy* untuk memperoleh *weight*/ bobot setiap *user flow* berdasarkan informasi yang diperoleh dari *rtt*, jumlah *hop* dan *loss ratio* per *user flow*:



Gambar 3.7 Proses Logika *Fuzzy* FTS

Data Pengguna/*User* (*INPUT* PREMIS/FAKTA berupa: *rtt, hop* dan *loss*) akan diolah didalam KOTAK HITAM dengan tujuan untuk memperoleh *OUTPUT* INFERENSI (bw).

Didalam KOTAK HITAM, FTS menggunakan fungsi implikasi yang mana tiap-tiap *rule* (aturan/proposisi) pada basis pengetahuan *fuzzy* akan berhubungan dengan suatu relasi. Dua fungsi implikasi yang digunakan adalah *min (minimum)* dan *dot (product)* [bab 2 fungsi implikasi hal. 18]:

- *Min (minimum)*. Fungsi ini akan memotong *output* himpunan *fuzzy*.
- *Dot (product)*. Fungsi ini akan menskala *output* himpunan *fuzzy*.

Sistem inferensi *fuzzy* yang mengacu pada metode Tsukamoto menjelaskan bahwa *output* hasil inferensi dari setiap aturan diberikan secara tegas (*crisp*) berdasarkan α-predikat (*fire strength*). Kemudian hasil akhirnya diperoleh dengan menggunakan rata-rata terbobot. Berikut penerapan FTS yang mengacu pada sistem inferensi *fuzzy* metode *Tsukamoto*:

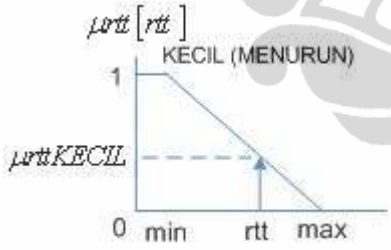
(Persamaan 3.1 *Rule*/aturan Logika *Fuzzy* FTS), potongan *rule* 1 dan 2 dijelaskan pada bab 2 sub bab fungsi implikasi, hal. 18. (Gambar keseluruhan pada gambar 3.8, hal. 41)

Tabel 3.2 *Rule* Logika *Fuzzy*

Condition (IF)			Impact (Then)
rtt	hop	loss	bw
KECIL	SEDIKIT	KECIL	NAIK
KECIL	SEDIKIT	BESAR	NAIK
KECIL	BANYAK	KECIL	NAIK
KECIL	BANYAK	BESAR	TURUN
BESAR	SEDIKIT	KECIL	NAIK
BESAR	BANYAK	KECIL	TURUN
BESAR	SEDIKIT	BESAR	TURUN
BESAR	BANYAK	BESAR	TURUN

Sebelum membahas *rule*/ aturan logika *fuzzy* FTS diatas, perlu diketahui bagaimana cara mencari μ **(derajat keanggotaan, rentang 0 s/d 1 (koordinat y *cartesian*))**. (*rtt*, *hop* dan *loss* sudah diketahui dari proses sniff.pl) dan *rttmax, rttmin, hopmax, hopmin, lossmax, lossmin* diperoleh dari nilai terendah dan tertinggi pengguna yang mengakses FTS *Server*.

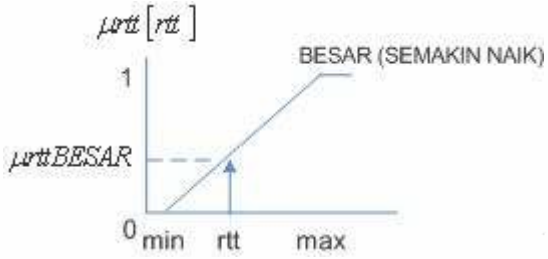
Langkah 1 (mencari μ derajat keanggotaan)



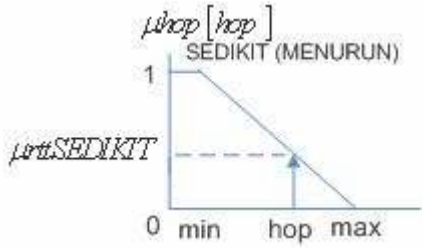
(Persamaan 3.2
Derajat keanggotaan $\mu rttKECIL[rtt]$*)*

$$\mu rttKECIL[rtt] = \frac{rtt\max - rtt}{rtt\max - rtt\min}$$

(Persamaan 3.3
Derajat keanggotaan $\mu rttBESAR[rtt]$*)*



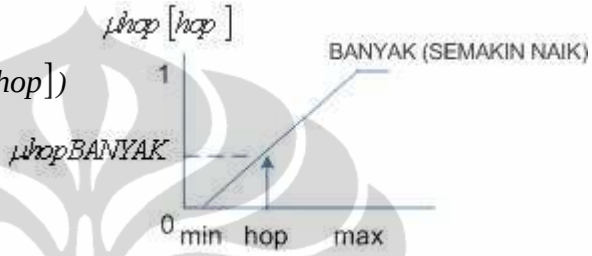
$$\mu rttBESAR[rtt] = \frac{rtt - rtt\min}{rtt\max - rtt\min}$$



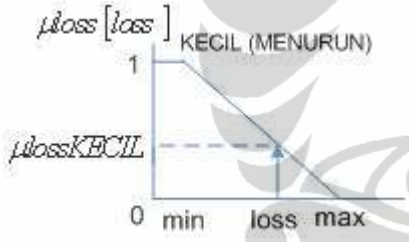
(Persamaan 3.4
Derajat keanggotaan $\mu hopSEDIKIT[hop]$*)*

$$\mu hopSEDIKIT[hop] = \frac{hop\max - hop}{hop\max - hop\min}$$

(Persamaan 3.5
Derajat keanggotaan $\mu hopBANYAK[hop]$*)*



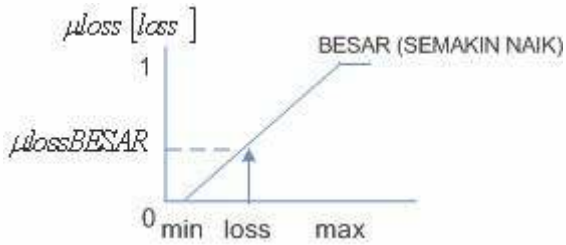
$$\mu hopBANYAK[hop] = \frac{hop - hop\min}{hop\max - hop\min}$$



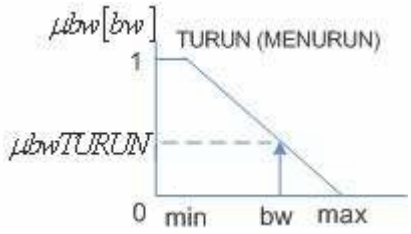
(Persamaan 3.6
Derajat keanggotaan $\mu lossKECIL[loss]$*)*

$$\mu lossKECIL[loss] = \frac{loss\max - loss}{loss\max - loss\min}$$

(Persamaan 3.7
Derajat keanggotaan $\mu lossBESAR[loss]$*)*



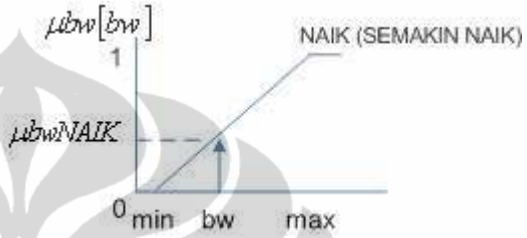
$$\mu lossBESAR[loss] = \frac{loss - loss\min}{loss\max - loss\min}$$



(Persamaan 3.8
Derajat keanggotaan $\mu bwTURUN[bw]$)

$$\mu bwTURUN[bw] = \frac{bw\max - bw}{bw\max - bw\min}$$

(Persamaan 3.9
Derajat keanggotaan $\mu bwNAIK[bw]$)



$$\mu bwNAIK[bw] = \frac{bw - bw\min}{bw\max - bw\min}$$

Langkah 2 (mencari α -predikat (*fire strength*)), disesuaikan dengan *rule* FTS dan fungsi implikasi *min (minimum)* / mencari harga *minimum*

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

α -predikat1 = $\mu rttKECIL \cap \mu hopSEDIKIT \cap \mu lossKECIL$

= min($\mu rttKECIL$, $\mu hopSEDIKIT$, $\mu lossKECIL$) *(Persamaan 3.10 α -predikat1)*

α -predikat1 menjadi harga $\mu bwNAIK$ (1)

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

α -predikat2 = $\mu rttKECIL \cap \mu hopSEDIKIT \cap \mu lossBESAR$

= min($\mu rttKECIL$, $\mu hopSEDIKIT$, $\mu lossBESAR$) *(Persamaan 3.11 α -predikat2)*

α -predikat2 menjadi harga $\mu bwNAIK$ (2)

Jika **rtt KECIL** dan **hop BANYAK** dan **loss KECIL**, maka **bw NAIK**

α -predikat3 = $\mu rttKECIL \cap \mu hopBANYAK \cap \mu lossKECIL$

= min($\mu rttKECIL$, $\mu hopBANYAK$, $\mu lossKECIL$) *(Persamaan 3.12 α -predikat3)*

α -predikat3 menjadi harga $\mu bwNAIK$ (3)

Jika **rtt KECIL** dan **hop BANYAK** dan **loss BESAR**, maka **bw TURUN**

α-predikat4 = $\mu rttKECIL \cap \mu hopBANYAK \cap \mu lossBESAR$

= min($\mu rttKECIL$, $\mu hopBANYAK$, $\mu lossBESAR$) *(Persamaan 3.13 α-predikat4)*

α-predikat4 menjadi harga $\mu bwTURUN$ (4)

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

α-predikat5 = $\mu rttBESAR \cap \mu hopSEDIKIT \cap \mu lossKECIL$

= min($\mu rttBESAR$, $\mu hopSEDIKIT$, $\mu lossKECIL$) *(Persamaan 3.14 α-predikat5)*

α-predikat5 menjadi harga $\mu bwNAIK$ (5)

Jika **rtt BESAR** dan **hop BANYAK** dan **loss KECIL**, maka **bw TURUN**

α-predikat6 = $\mu rttBESAR \cap \mu hopBANYAK \cap \mu lossKECIL$

= min($\mu rttBESAR$, $\mu hopBANYAK$, $\mu lossKECIL$) *(Persamaan 3.15 α-predikat6)*

α-predikat6 menjadi harga $\mu bwTURUN$ (6)

Jika **rtt BESAR** dan **hop SEDIKIT** dan **loss BESAR**, maka **bw TURUN**

α-predikat7 = $\mu rttBESAR \cap \mu hopSEDIKIT \cap \mu lossBESAR$

= min($\mu rttBESAR$, $\mu hopSEDIKIT$, $\mu lossBESAR$) *(Persamaan 3.16 α-predikat7)*

α-predikat7 menjadi harga $\mu bwTURUN$ (7)

Jika **rtt BESAR** dan **hop BANYAK** dan **loss BESAR**, maka **bw TURUN**

α-predikat8 = $\mu rttBESAR \cap \mu hopBANYAK \cap \mu lossBESAR$

= min($\mu rttBESAR$, $\mu hopBANYAK$, $\mu lossBESAR$) *(Persamaan 3.17 α-predikat8)*

α-predikat8 menjadi harga $\mu bwTURUN$ (8)

Langkah 3 (mencari skala *output bandwidth*), sesuai dengan persamaan aritmatika untuk $\mu bwTURUN[bw]$ dan $\mu bwNAIK[bw]$ dimana harga $\mu bwNAIK$ dan $\mu bwTURUN$ sudah diketahui dari harga α-predikat masing-masing *rule*, maka dimasukkan ke persamaan (3.8 dan 3.9).

$$\mu bwTURUN[bw] = \frac{bw\max - bw}{bw\max - bw\min}$$

dan

$$\mu bwNAIK[bw] = \frac{bw - bw\min}{bw\max - bw\min}$$

Sehingga, setiap aturan memperoleh harga bw masing-masing (dengan bwmax dan bwmin, dimasukkan manual dalam pertama kali *load* hit.pl)

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

bw1 = bwmin + (α-predikat1 * (bwmax - bwmin)) *(Persamaan 3.18 bw1)*

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

bw2 = bwmin + (α-predikat2 * (bwmax - bwmin)) *(Persamaan 3.19 bw2)*

Jika **rtt KECIL** dan **hop BANYAK** dan **loss KECIL**, maka **bw NAIK**

bw3 = bwmin + (α-predikat3 * (bwmax - bwmin)) *(Persamaan 3.20 bw3)*

Jika **rtt KECIL** dan **hop BANYAK** dan **loss BESAR**, maka **bw TURUN**

bw4 = bwmax - (α-predikat4 * (bwmax - bwmin)) *(Persamaan 3.21 bw4)*

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

bw5 = bwmin + (α-predikat5 * (bwmax - bwmin)) *(Persamaan 3.22 bw5)*

Jika **rtt BESAR** dan **hop BANYAK** dan **loss KECIL**, maka **bw TURUN**

bw6 = bwmax - (α-predikat6 * (bwmax - bwmin)) *(Persamaan 3.23 bw6)*

Jika **rtt BESAR** dan **hop SEDIKIT** dan **loss BESAR**, maka **bw TURUN**

bw7 = bwmax - (α-predikat7 * (bwmax - bwmin)) *(Persamaan 3.24 bw7)*

Jika **rtt BESAR** dan **hop SEDIKIT** dan **loss BESAR**, maka **bw TURUN**

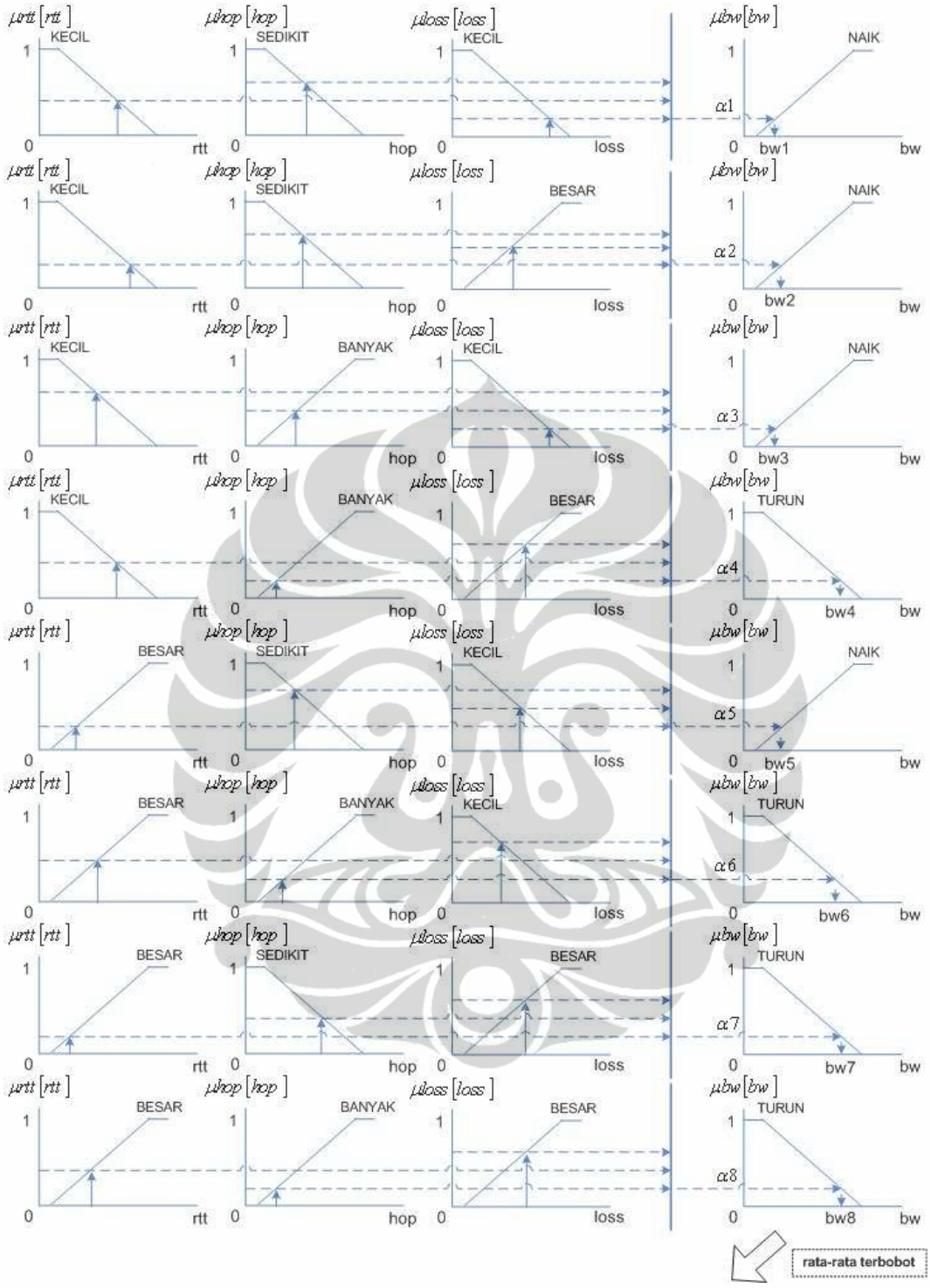
bw8 = bwmax - (α-predikat8 * (bwmax - bwmin)) *(Persamaan 3.25 bw8)*

Langkah 4 (mencari rata-rata terbobot), yang merupakan langkah terakhir dalam perhitungan logika *fuzzy* dengan metode Tsukamoto, karena metode *dot (product)* sudah diterapkan disini.

$$bw\_avg = \frac{\alpha 1 bw1 + \alpha 2 bw2 + \alpha 3 bw3 + \alpha 4 bw4 + \alpha 5 bw5 + \alpha 6 bw6 + \alpha 7 bw7 + \alpha 8 bw8}{\alpha 1 + \alpha 2 + \alpha 3 + \alpha 4 + \alpha 5 + \alpha 6 + \alpha 7 + \alpha 8}$$

(Persamaan 3.26 bv_avg (rata-rata terbobot))

Keterangan selengkapnya ada pada gambar 3.8.



Gambar 3.8 Inferensi Lengkap FTS dengan Metode *Tsukamoto*

Rata-rata Terbobot (Pada Gambar diatas):

$$bw\_avg = \frac{\alpha 1 bw1 + \alpha 2 bw2 + \alpha 3 bw3 + \alpha 4 bw4 + \alpha 5 bw5 + \alpha 6 bw6 + \alpha 7 bw7 + \alpha 8 bw8}{\alpha 1 + \alpha 2 + \alpha 3 + \alpha 4 + \alpha 5 + \alpha 6 + \alpha 7 + \alpha 8}$$

Pada Gambar diatas menjelaskan bahwa pengambilan Nilai *input* yang *minimum* dari (*rtt, hop* dan *loss*) (3 Bagian grafik sebelah kiri), akan melakukan pemotongan secara *minimum* dan menjadi α-predikat (*fire strength*) *output* (1 bagian grafik sebelah kanan) yang kemudian akan menjadi salah satu *input* untuk memperoleh nilai bw setiap aturan tersebut. Jumlah perkalian antara seluruh (α-predikat dengan bw-nya) akan dibagi dengan jumlah α predikat setiap aturan sehingga diperoleh rata-rata terbobot.

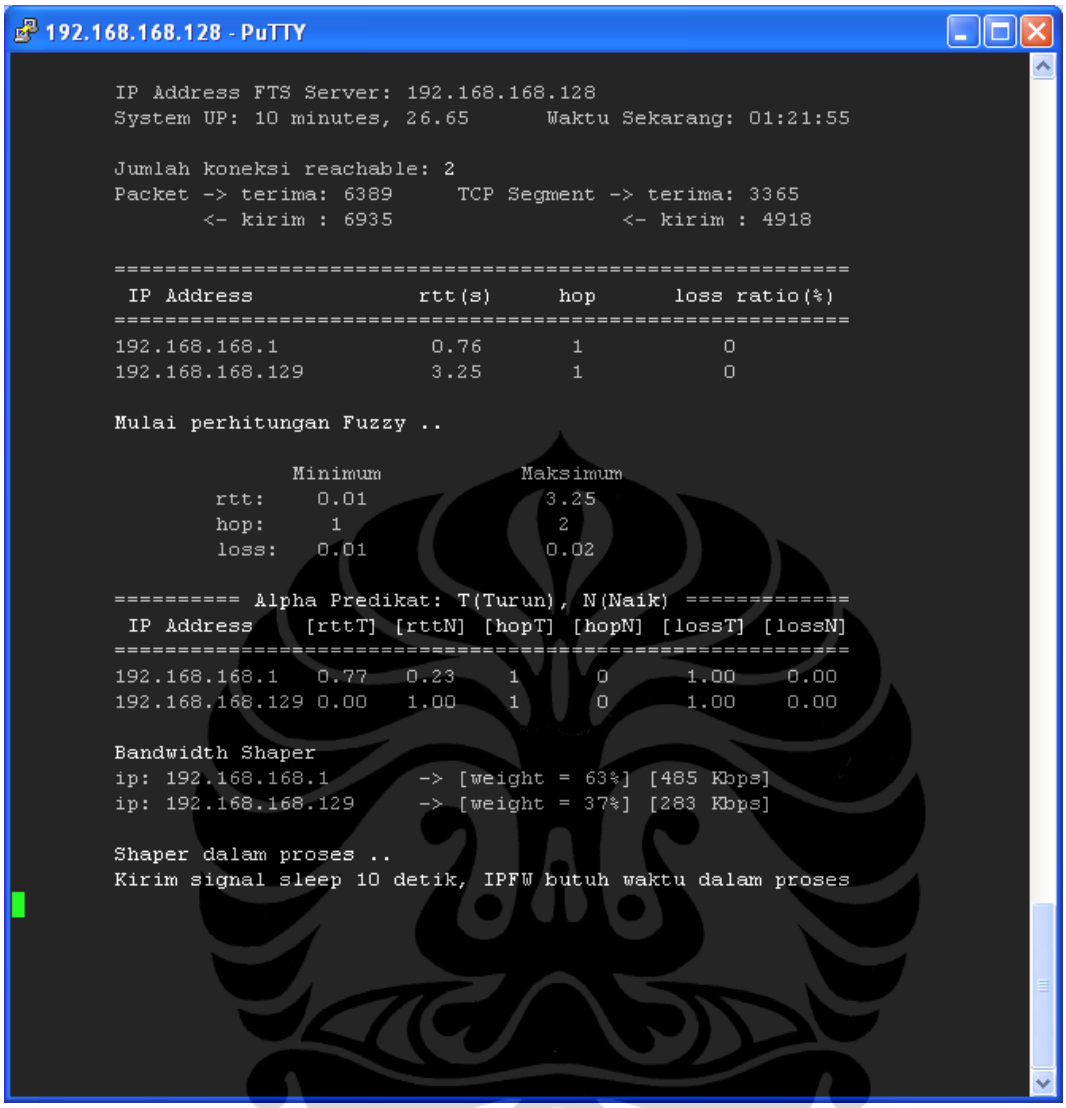
Setelah melalui proses logika *fuzzy*, maka dicari alokasi *weight* tiap pengguna:

$$bwperuser = \frac{bw\_avg[user(n)]}{total[bw\_avg]} x(bw\max)$$

(Persamaan 3.27 Mencari bw setiap pengguna)

$$weight = \frac{bwperuser}{bw\max} x100\%$$

(Persamaan 3.28 Mencari weight setiap pengguna)



Gambar 3.9 Tampilan hit.pl

3.4 *Traffic Monitoring*

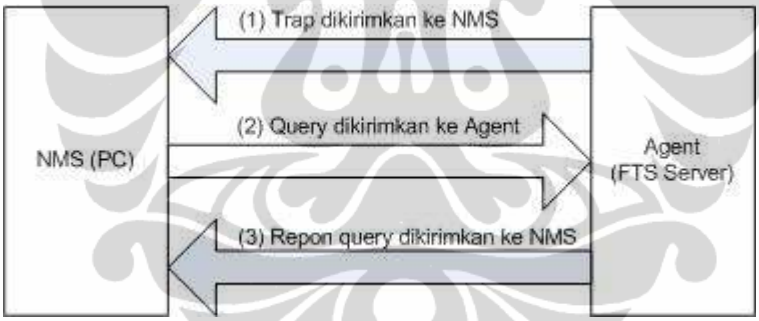
Bandwidth traffic monitoring akan menggunakan *SNMP Traffic Grapher* (STG) (berbasis Windows) dengan mengambil *Object Identifier* (OID) tertentu pada *daemon* SNMP yang diinstal disisi *server* (UNIX). Versi SNMP yang digunakan adalah versi 1 (SNMPv1) yang mengacu pada RFC 1157 yang terdiri dari 3 hak akses *community*: *read-only*, *read-write* dan *trap*. Dengan alasan keamanan sistem, maka yang digunakan adalah hak akses *read-only* yang hanya melakukan proses baca informasi dari *server*, tanpa bisa merubah apapun konfigurasi di *server* secara *remote*.

3.4.1 *Manager* dan *Agent*

Berikut akan diulas secara singkat mengenai koneksi SNMP yang akan dipergunakan di FTS. Pada protocol SNMP (UDP): SNMPv1 (RFC 1157), SNMPv2 (RFC 1905-1907) dan SNMPv3 (RFC 1906-1907 dan RFC 2571-2575) pasti dikenal dengan istilah *Manager* dan A*gent*. *Manager* adalah PC yang menjalankan *software* SNMP *client* (bukan SNMP *Daemon*) yang akan mengambil informasi *resource* sebuah *Agent*. Sekumpulan *Manager* disebut dengan *Network Management Stations (NMSs)*. Satu NMS melakukan *polling* dan menerima *trap* dari *Agent*.

Pada laporan tesis ini, satu *client* pengontrol (PC) akan bertugas sebagai NMS yang akan melakukan *polling* ke *Agent* (FTS *Server*). *Software snmp agent* yang akan di*install* pada *Agent* (FTS *server*) adalah ucd-snmp.

Berikut proses yang dijalankan antara NMS dengan *Agent*:



Gambar 3.10 Hubungan antara NMS dengan *Agent* [15]

Detail proses:

1. FTS *Server* mengirimkan sinyal *trap* ke NMS.
2. NMS melakukan *polling (query)* OID tertentu ke *Agent*.
3. *Agent* akan merespon balik dengan mengirimkan informasi OID yang diminta NMS.

3.4.2 *Management Information Base* (MIB)-II

Berikut akan diulas secara singkat mengenai OID yang akan diambil oleh NMS. Terlebih dahulu akan diulas mengenai MIB. MIB merupakan sebuah

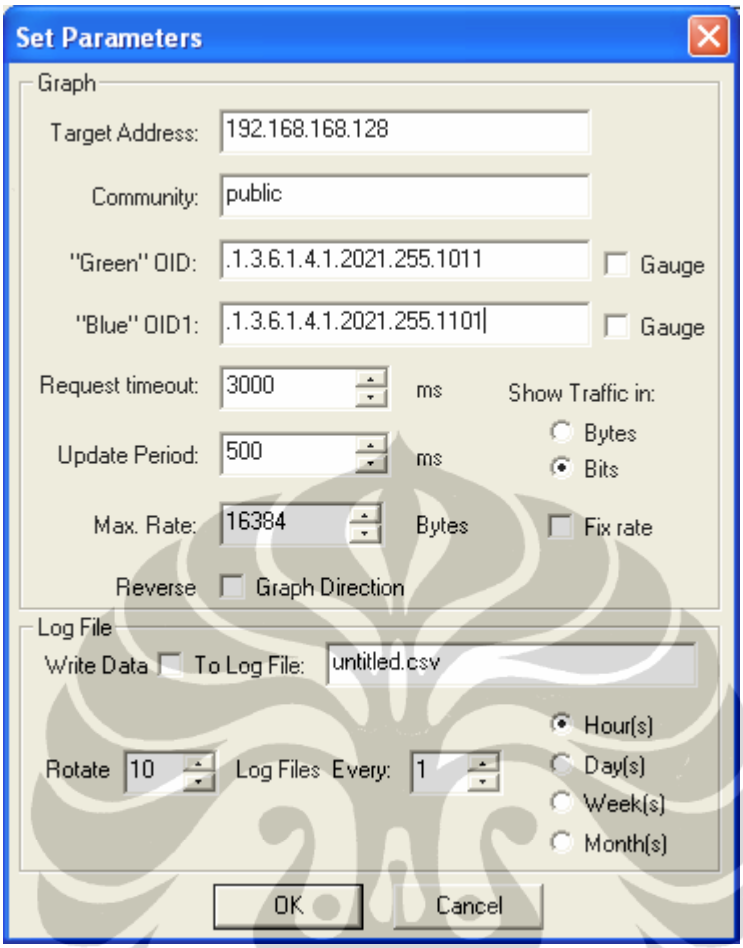
database object manajemen yang terdiri dari statistik informasi yang dipunyai oleh sebuah *Agent* apakah itu status *resource* CPU, *memory* RAM, kecepatan *interface*, MTU, *octet* yang dikirimkan/ diterima, lokasi sistem, kontak informasi, dsb. Mengenai MIB-II diulas di RFC 1213.

Laporan tesis ini tidak membahas seluruh *object* didalam OID yang mengacu pada *Structure of Management Information (SMI)*, karena dapat mencapai ribuan informasi yang dipunyai oleh sebuah *Agent*. Sejak *Corporate Enterprise* diperbolehkan untuk meng-*create* sendiri OID tertentu, maka penulis merancang *octet* terakhir dari OID dengan menyesuaikan *rule queue* IPFW (*traffic shaper*) untuk mengetahui besar *bandwidth* yang dialokasikan ke setiap pengguna oleh FTS dengan acuan *substree name* "*interfaces*" pada SMI.

Tabel 3.3 OID *bandwidth monitoring* di FTS

Daftar Pengguna	OID *bandwidth downstream*	OID *bandwidth upstream*
Pengguna 1	.1.3.6.1.4.1.2021.255.**1011**	.1.3.6.1.4.1.2021.255.**1101**
Pengguna 2	.1.3.6.1.4.1.2021.255.**1012**	.1.3.6.1.4.1.2021.255.**1102**
Pengguna n	.1.3.6.1.4.1.2021.255.**101n**	.1.3.6.1.4.1.2021.255.**110n**

Pada tabel 3.2 baris 2 (Pengguna 1), mempunyai nilai bagian *octet* terakhir untuk OID *bandwidth downstream* = 1011 dan OID *bandwidth upstream* = 1101. Nilai *octet* terakhir tersebut akan terus bertambah (*increase* 1) setiap ada pengguna baru yang melakukan akses ke FTS. Nilai tersebut merupakan nomor *rule queue* IPFW yang diterapkan pada *sharing bandwidth* menggunakan WF2Q+ di FTS. Untuk memperoleh *monitoring* yang *realtime*, maka *update period* di-*set* 500ms. Berikut konfigurasi STG yang mengacu pada OID Pengguna 1:



Gambar 3.11 Konfigurasi STG

Gambar 3.12 Contoh tampilan STG yang mengacu pada *queue* IPFW