ANALISIS CLASS BASED QUEUING (CBQ) DAN HIERARCHICAL TOKEN BUCKET (HTB) UNTUK MANAJEMEN BANDWIDTH PADA JARINGAN TCP/IP

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan

Memperoleh gelar Sarjana Teknik

Disusun oleh :

ADI PRATOMO

NIM. 0001060268

DEPARTEMEN PENDIDIKAN NASIONAL

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2007

ANALISIS CLASS BASED QUEUING (CBQ) DAN HIERARCHICAL TOKEN BUCKET (HTB) UNTUK MANAJEMEN BANDWIDTH PADA JARINGAN TCP/IP

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan

Memperoleh gelar Sarjana Teknik

Disusun oleh :

ADI PRATOMO

NIM. 0001060268

Telah Diperiksa Dan Disetujui Oleh

DOSEN PEMBIMBING

Raden Arif Setiawan, ST, MT.	Ir. Primantara Hari Trisnawan
NIP. 132 231 713	NIP. 132 090 390

KATA PENGANTAR

Puji syukur kami panjatkan kehadirat Tuhan Yang Maha Esa, yang telah melimpahkan rahmat-Nya sehingga penulis bisa menyelesaikan skripsi ini dengan baik dan tepat pada waktunya.

Pada kesempatan ini, penulis ingin menyampaikan terima kasih kepada :

1. Dosen pembimbing yang telah membimbing dan mengarahkan serta memberikan wejangan kepada penulis selama penulisan skripsi ini.
2. Bapak Kapala Jurusan dan Bapak Sekertaris Jurusan, Jurusan Teknik Elektro yang membantu penulis dalam penulisan skripsi dan mengarahkan penulis selama di kampus.
3. Teman- teman yang telah membantu penulis selama penulisan skripsi ini.

Penulis menyadari bahwa skripsi ini belum sempurna dan masih ada kekurangan sehingga penulis mengharapkan kritik dan saran untuk kesempurnaan skripsi ini.

Akhirnya semoga laporan ini bisa mamberikan manfaat bagi kita semua dan kita mampu mengambil manfaatnya.

Malang, 6 Agustus 2007

Penulis

ABSTRAK

ADI PRATOMO. 2007 : Analisis Class Based Queuing (CBQ) Dan Hierarchical Token Bucket (HTB) Untuk Manajemen Bandwidth Pada Jaringan TCP/IP. Skripsi Jurusan Teknik Elektro, Fakultas Tekink, Universitas Brawijaya. Pembimbing : Raden Arif Setiawan, ST. MT. dan Ir. Primantara Hari Trisnawan.

Internet telah memberikan pengaruh yang sangat besar pada penyebaran informasi berupa data, dengan demikian akan semakin banyak individu yang mengakses data melalui Internet. *Bandwidth manager* memegang perananan penting dalam mengatur jenis aplikasi yang bisa mengakses *link* yang ada. Komponen paling penting dalam *bandwidth manager* adalah *queuing disciplines*. Dua di antaranya adalah dengan menggunakan teknik *Class Based Queuing (CBQ)* dan *Hierarchical Token Bucket (HTB)*.

Kedua disiplin antrian ini akan diimplementasikan dan dibandingkan berdasarkan parameter waktu proses, *throughput*, *jitter*, dan *loss datagram* dengan pembedaan pada pembagian kapasitas *bandwidth*, nomor *port* dan alamat IP *address*.

Dari hasil pengujian didapatkan hasil bahwa *throughput* yang dihasilkan kelas yang menggunakan HTB secara umum mampu memenuhi ketentuan/*rule* yang diimplementasikan dibanding CBQ. Hal ini berarti CBQ kurang akurat dibanding HTB dalam hal *throughput* yang dihasilkan pada setiap kelas. CBQ memberikan nilai *jitter* sampai dengan 92,17 % lebih kecil dibanding dengan HTB. CBQ memberikan waktu proses sampai dengan 124,52 % lebih cepat dibanding dengan HTB. CBQ memberikan *loss datagram* sampai 58,60 % sedangkan HTB memberikan nilai yang stabil pada *loss datagram*, yaitu nyaris tidak menghasilkan nilai *loss datagram* kecuali pada skenario dengan 4 *leaf class* itupun dengan nilai *loss* yang sangat kecil sekali 5 %.

Kata Kunci : *bandwidth manajer*, HTB, CBQ, *link sharing*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR ISTILAH

Jaringan komputer :

sekelompok komputer otonom yang saling berhubungan antara satu dengan lainnya menggunakan protokol komunikasi melalui media komunikasi sehingga dapat saling berbagi sumber daya yang ada.

Internet (dengan "I" besar) :

sistem komputer umum, yang berhubungan secara global dan menggunakan TCP/IP sebagai protokol pertukaran paket data (packet switching communication protocol).

internet (dengan "i" kecil) :

rangkaian komputer yang saling berhubungan membentuk suatu jaringan komputer yang besar.

Gateway :

perangkat untuk interkoneksi jaringan dimana masing - masing jaringan memiliki protokol yang berbeda.

Link :

jalur atau kanal dimana data ditransmisikan.

Node :

lokasi pemrosesan. Sebuah *node* dapat berupa sebuah komputer atau perangkat lain, seperti printer. Setiap *node* memiliki MAC *address* yang unik.

Hop :

satuan langkah, dari satu router ke router berikutnya, yang dilewati paket data untuk menuju alamat tujuan.

Delay :

merupakan total waktu yang dilalui suatu paket dari pengirim ke penerima melalui jaringan.

Jitter :

merupakan variasi dari *delay* end-to-end.

Bandwidth :

merupakan *rate* transfer data maksimal yang dapat diteruskan antara dua titik.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan komunikasi data telah mencapai kemajuan yang sangat pesat, ditandai oleh pemakaiannya yang begitu meluas tidak hanya oleh perusahaan-perusahaan berskala besar tetapi juga oleh individu-individu yang membutuhkan informasi yang cepat dan terbarukan melalui jaringan komunikasi data terutama melalui Internet[1]. Internet telah memberikan pengaruh yang sangat besar pada penyebaran informasi berupa data, dengan demikian akan semakin banyak individu yang mengakses data melalui Internet. Hal ini akan berakibat pada meningkatnya trafik data yang dapat menyebabkan penurunan performa jaringan, contohnya terjadi penurunan *bandwidth* yang diterima *user* atau *user* tidak dapat terhubung ke server yang dituju, terutama pada jaringan yang memiliki *bandwidth* terbatas dan yang tidak berkeinginan untuk menambah kapasitas *bandwidth*-nya (*bandwidth* berbanding lurus dengan biaya). Untuk mengurangi penurunan performansi jaringan tanpa menambah biaya (atau *bandwidth*) salah satunya dengan menerapkan manajemen *bandwidth* pada *gateway*[2].

Manajemen *bandwidth* menjadi hal yang mutlak diperlukan bagi jaringan multi layanan, semakin banyak dan bervariasinya aplikasi yang dapat dilayani oleh suatu jaringan berpengaruh pada penggunaan *link*[3] dalam jaringan tersebut. *Link-link* yang ada harus mampu menangani kebutuhan *user* akan aplikasi tesebut bahkan dalam keadaan *kongesti*[4] sekalipun, harus ada suatu jaminan bahwa *link* tetap dapat berfungsi sebagaimana mestinya walaupun terjadi ledakan permintaan layanan.

---

[1] Internet (dengan "I" besar) adalah sistem komputer umum, yang berhubungan secara global dan menggunakan TCP/IP sebagai protokol pertukaran paket data (packet switching communication protocol).

[2] *Gateway* adalah perangkat untuk interkoneksi jaringan dimana masing - masing jaringan memiliki protokol yang berbeda [FOR-01].

[3] *Link* adalah jalur atau kanal dimana data ditransmisikan.

[4] Kongesti adalah situasi dimana terlalu banyak paket yang berada di dalam jaringan yang menyebabkan performa jaringan menurun [TAN-03].

Bandwidth manager memegang perananan penting dalam mengatur jenis aplikasi yang bisa mengakses *link* yang ada. Bahkan dalam keadaan tertentu ketika alokasi *bandwidth* yang dimiliki oleh suatu aplikasi/layanan tidak digunakan maka oleh *bandwidth manager* alokasi *bandwidth* yang *idle* tersebut dapat dialihkan sementara waktu kepada kelas yang sedang mengalami *backlog*/timbunan antrian. Hal ini memberikan keuntungan mempercepat hilangnya *backlog* suatu kelas sekaligus mengoptimalkan penggunaan *link* yang ada.

Komponen paling penting dalam *bandwidth manager* adalah *queuing disciplines*. *Queuing disciplines* (disiplin antrian) adalah algoritma yang menentukan bagaimana antrian paket akan dikirimkan [ALM-01]. Dengan mengatur bagaimana antrian paket akan dikirimkan maka kita dapat melakukan pengaturan *bandwidth*.

Ada berbagai macam disiplin antrian yang dapat digunakan untuk manajemen *bandwidth*. Salah satunya adalah *Class Based Queuing* (CBQ). CBQ merupakan sebuah disiplin antrian yang memberikan pembagian *bandwidth* antara *user* yang menggunakan *link* fisik yang sama. Salah satu kelebihan dari CBQ adalah kemampuan menerapkan pembagian *bandwidth* melalui struktur kelas-kelas[5] secara hirarki. Setiap kelas dapat memiliki atribut yang berbeda, contoh berdasarkan nomor *port* dan/atau alamat IP *address*. Setiap kelas memiliki antriannya masing-masing dan diberikan jatah *bandwidth*-nya. Sebuah kelas *child*[6] dapat meminjam *bandwidth* dari kelas *parent*[7] selama terdapat kelebihan *bandwidth* [FLO-95]. CBQ merupakan disiplin antrian yang paling tua dan paling banyak digunakan. CBQ telah diimplementasikan di berbagai platform, antara lain Linux [HUB-02], Solaris [WAK-95][HOF-94] dan FreeBSD [KCH-98].

CBQ bukan merupakan disiplin antrian yang sempurna. CBQ merupakan disiplin antrian yang paling rumit dan tidak cocok dengan cara kerja sistem operasi Linux [HUB-02]. Martin Devera menyadari bahwa CBQ sangat komplek dan tidak dioptimalkan untuk berbagai macam situasi yang umum. Martin Devera memperkenalkan suatu disiplin antrian baru untuk sistem operasi Linux. Disiplin

[5] Kelas adalah penggolongan trafik berdasarkan *network*, alamat IP *address*, dan/atau nomor *port*.
[6] Kelas *child* adalah kelas yang merupakan bagian dari kelas yang berada di hirarki diatasnya.
[7] Kelas *parent* adalah kelas yang merupakan induk dari kelas-kelas yang berada di hirarki dibawahnya.

antrian ini menggunakan Token Bucket Filter yang memiliki kelas. Martin Devera menyebut disiplin antrian ini *Hierarchical Token Bucket* (HTB) [DEV-02][DEV-03].

Dalam penelitian tugas akhir ini akan dilakukan analisis *bandwidth manager* menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB) pada sistem operasi Linux. Analisis dilakukan berdasarkan parameter waktu proses, *throughput*[8], *jitter*[9], dan *loss datagram* [10]dengan pembedaan pada pembagian kapasitas *bandwidth*, nomor *port* dan alamat IP *address*. Dari analisis ini diharapkan didapat jenis disiplin antrian yang paling baik dan optimal diantara keduannya yang bisa secara langsung diimplementasikan pada komputer *gateway* berbasis sistem operasi Linux.

1.2 Rumusan Masalah

Berdasarkan pada permasalahan yang telah dijelaskan pada bagian latar belakang, maka rumusan masalah dikhususkan pada:

1. Merancang dan mengkonfigurasi Linux agar dapat berfungsi sebagai *bandwidth manager* menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB)
2. Merancang dan mengkonfigurasi Linux agar dapat berfungsi sebagai *bandwidth manager* berdasarkan pada kapasitas *bandwidth*, nomor *port* dan alamat IP *address*.
3. Menganalisis dan memonitoring jalannya paket data yang melewati komputer *bandwidth manager*.
4. Mengukur kinerja dan kemampuan *bandwidth manager* berdasarkan parameter waktu proses, *throughput*, *jitter*, dan *loss datagram* dengan pembedaan pada pembagian kapasitas *bandwidth*, nomor *port* dan alamat IP *address*.

[8] *Throughput* adalah jumlah data yang terkirim melalui suatu *link* tertentu per satuan waktu.
[9] *Jitter* adalah variasi dari *delay* end-to-end.
[10] *loss datagram* adalah jumlah paket data yang tidak diterima oleh penerima atau hilang dalam pengiriman

1.3 Batasan Masalah

Dalam perencanaan dan pembuatan skripsi ini perlu dilakukan pembatasan masalah. Pembatasan masalah yang diajukan dalam skripsi ini antara lain:

1. Masalah yang akan diteliti adalah melakukan analisis implementasi *bandwidth manager* menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB).
2. Analisis dilakukan berdasarkan pada kapasitas *bandwidth*, nomor *port* dan alamat IP *address* pada jaringan berbasis TCP/IP.
3. Perangkat lunak yang digunakan untuk manajemen *bandwidth* pada *bandwidth manager* adalah kernel Linux.
4. Sistem operasi yang digunakan pada perangkat *bandwidth manager* adalah Linux distribusi Slackware versi 11 dan program-program yang digunakan adalah program dengan standar Linux distribusi Slackware versi 11.

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah melakukan analisis implementasi *bandwidth manager* menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan *Hierarchical Token Bucket* (HTB) berdasarkan parameter waktu proses, *throughput*, *jitter*, dan *loss datagram*. dengan pembedaan pada pembagian kapasitas *bandwidth*, nomor *port* dan alamat IP *address*, sehingga diharapkan didapat jenis disiplin antrian yang paling baik dan optimal diantara keduanya yang bisa secara langsung diimplementasikan pada komputer *gateway* berbasis sistem operasi Linux.

1.5 Manfaat

Manfaat yang bisa didapatkan dari tugas akhir ini adalah :

1. Sebagai sumber referensi dalam pemilihan *bandwidth manager* yang tepat untuk diterapkan pada berbagai macam situasi.

2. Memberikan panduan bagi masyarakat dalam mengatur *bandwidth* pada jaringan berbasis TCP/IP sehingga dapat digunakan secara optimal dan efisien.

1.6. Sistematika Pembahasan

Sistematika penulisan dalam skripsi ini sebagai berikut:

BAB I Pendahuluan

Memuat latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika pembahasan.

BAB II Dasar Teori

Membahas teori dasar dan teori penunjang mengenai Konsep Jaringan Komputer, protokol TCP/IP, *Quality of Service (QoS)*, *Linux Traffic Control* dan Disiplin Antrian.

BAB III Metode Penelitian

Membahas tentang metode yang digunakan dalam penulisan yang terdiri dari studi literatur, perancangan dan pembuatan, pengujian dan analisis, serta pengambilan kesimpulan dan saran.

BAB IV Perancangan

Membahas perancangan *bandwidth manager* pada jaringan komputer menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan *Hierarchical Token Bucket* (HTB).

BAB V Implementasi

Membahas implementasi *bandwidth manager* pada jaringan komputer menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan *Hierarchical Token Bucket* (HTB).

BAB VI Pengujian dan Analisis

Memuat hasil pengujian dan analisis terhadap sistem yang telah direalisasikan.

BAB VII Kesimpulan dan Saran

Memuat kesimpulan yang diperoleh dari pengujian dan analisis sistem, serta saran-saran untuk pengembangan lebih lanjut.

BAB II

DASAR TEORI

Bab ini akan menjelaskan tinjauan pustaka dan dasar teori yang menunjang penulisan skripsi mengenai analisis *Class Based Queuing* (CBQ) dan *Hierarchical Token Bucket* (HTB) untuk manajemen *bandwidth* pada jaringan TCP/IP.

Dasar teori yang digunakan adalah teori konsep jaringan komputer , protokol TCP/IP, *Quality of Service (QoS)*, *Linux Traffic Control* dan *Queuing Disciplines*.

2.1 Konsep Jaringan Komputer

Jaringan komputer adalah sekelompok komputer otonom yang saling berhubungan antara satu dengan lainnya menggunakan protokol komunikasi melalui media komunikasi sehingga dapat saling berbagi sumber daya yang ada [TIM-03]. Istilah – istilah yang berkaitan dengan jaringan komputer umumnya *gateway, router, hub* dan *switch*. *Gateway* adalah perangkat untuk interkoneksi jaringan di mana masing - masing jaringan memiliki protokol yang berbeda [FOR-01]. *Router* merupakan perangkat untuk menghubungkan dua atau lebih *network* dan bertugas sebagai perantara dalam menyampaikan data antar jaringan komputer [LAM-04]. *Hub* merupakan *repeater* dengan banyak *port*[11]. Pada alat ini ketika sinyal digital elektronik diterima pada sebuah *port*, sinyal tersebut akan dikuatkan kembali atau dihasilkan ulang dan dikirimkan ke semua segmen kecuali segmen dari mana sinyal tersebut diterima [LAM-04]. *Switch* adalah perangkat dengan sebuah miniatur *bridge*[12] di dalam setiap portnya. Perangkat tersebut dapat melacak alamat - alamat *Media Access Control (*MAC *address[13])* yang terhubung ke setiap *port*nya dan melakukan *route* lalu lintas *network* yang ditujukan ke alamat MAC tertentu hanya kepada *port* di mana alamat itu berada [BRE-03].

[11] *Port* (hardware) adalah sebuah *interface* antara komputer dengan komputer lain atau perangkat lain.

[12] *Bridge* adalah sebuah kotak kecil dengan dua buah konektor *network* yang terhubung ke dua bagian yang terpisah dari *network.* Bridge memiliki fungsi dari hub (penguatan sinyal), tetapi *bridge* bisa mengenali *frame* dari data. Bridge memiliki sebuah tabel, mencatat alamat MAC yang bisa diakses secara langsung oleh setiap portnya. Bridge kemudian menggunakan informasi tersebut untuk mengontrol dan mengatur aliran data di *network* [BRE-03].

[13] MAC *address* adalah sebuah alamat *hardware* yang diperlukan oleh semua *port* atau semua alat untuk terhubung ke sebuah segmen LAN(*Local Area Network*)

Jarak merupakan hal yang penting dalam membangun sebuah jaringan komputer, karena untuk setiap jarak yang berbeda diperlukan teknik yang berbeda-beda pula. Berdasarkan jarak dan area kerjanya jaringan komputer dibedakan menjadi tiga kelompok yaitu: *LAN (Local Area Network), MAN (Metropolitan Area Network), dan WAN (Wide Area Network)* [TIM-03].

2.1.1 *Local Area Network* (LAN)

LAN adalah jaringan yang digunakan untuk menghubungkan komputer – komputer pribadi dan *workstation* dalam suatu perusahaan yang menggunakan peralatan secara bersama-sama dan saling bertukar informasi [TIM-03]. LAN dimiliki oleh perusahaan tanpa menggunakan fasilitas dari perusahaan telekomunikasi umum. LAN dibangun menggunakan topologi terentu. Topologi adalah istilah yang digunakan untuk menguraikan cara bagaimana komputer terhubung dalam suatu jaringan. Topologi pada LAN ada lima yaitu *Linear Bus*, *Ring*, *Star, Tree,* dan *Mesh* [TIM-03].

2.1.1.1 *Linear Bus*

Topologi *linear bus* terdiri dari satu kabel utama dengan sebuah *terminator*[14] pada tiap ujungnya. Setiap *node*[15] terkoneksi pada kabel utama [BUD-02]. Pada topologi jenis ini semua terminal terhubung ke jalur komunikasi. Informasi yang dikirimkan akan melewati semua terminal pada jalur tersebut. Pada saat alamat yang tercantum dalam data atau informasi yang dikirim sesuai dengan alamat terminal yang dilewati, data atau informasi tersebut akan diterima dan diproses. Pada saat alamat tersebut tidak sesuai, maka informasi tersebut akan diabaikan oleh terminal yang dilewati [TIM-03]. Skema topologi *Linear Bus* dapat dilihat dalam Gambar 2.1.

Keuntungan :

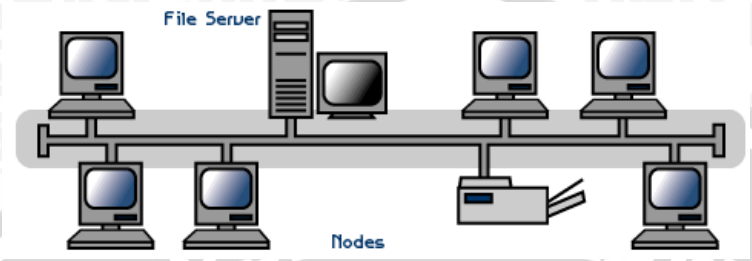
Hemat kabel, Layout kabel sederhana, mudah dikembangkan, tidak butuh kendali pusat

[14] *Terminator* adalah perangkat yang memberikan resistensi elektronik pada ujung dari kabel transmisi [HSC-04]

[15] *Node* adalah lokasi pemrosesan. Sebuah *node* dapat berupa sebuah komputer atau perangkat lain, seperti printer. Setiap *node* memiliki MAC *address* yang unik.

Kerugian:

Sulit dalam mengisolasi kesalahan jaringan, kepadatan lalulintas tinggi, kecepatan transfer data akan menurun bila jumlah pemakai bertambah, diperlukan *repeater* untuk jarak jauh



Gambar 2.1 Topologi *Linear Bus*

Sumber: [FCI-05]

2.1.1.2 *Ring*

Topologi *ring* mirip dengan topologi *bus,* tetapi kedua terminal yang berada di ujung saling dihubungkan [TIM-03]. Topologi ini menyerupai lingkaran. Komputer yang akan mengirim data harus menunggu token[16] sampai ke tempatnya kemudian meng*attach* data pada token dan mengembalikan token dan data tersebut pada jaringan [MAR-06]. Pada saat token mencapai tujuan yang diinginkan, komputer penerima akan mengambil data dari token. Token dikembalikan ke jaringan sehingga proses pengiriman data pada komputer lain dapat dimulai [MAR-06]. Setiap terminal dalam jaringan ini saling tergantung, sehingga jika terjadi kerusakan pada satu terminal maka seluruh jaringan akan terganggu [TIM-03] Skema topologi *ring* dapat dilihat dalam Gambar 2.2.

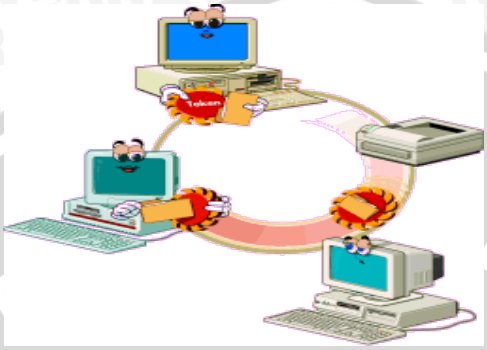
Keuntungan :

Hemat kabel, dapat melayani lalulintas data yang padat

[16] Token adalah paket khusus yang berisi data dan bertindak sebagai pembawa pesan pada tiap komputer dan perangkat pada topologi ring [CIT-04]

Kerugian :

Peka kesalahan, pengembangan jaringan lebih kaku, kerusakan pada media pengirim / terminal dapat melumpuhkan kerja seluruh jaringan, lambat karena pengiriman menunggu giliran token.

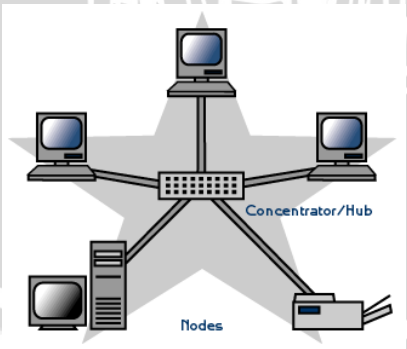


Gambar 2.2 Topologi *Ring*

Sumber: [MIL-05]

2.1.1.3 *Star*

Pada topologi *star* setiap terminal terkoneksi secara langsung ke *central network hub* atau *concentrator* [TIM-03]. Pengiriman data dari satu terminal lain ke terminal lainnya melalui *central network hub* atau *concentrator*. *Hub* atau *concentrator* bertindak sebagai pengatur dan pengendali semua komunikasi data yang terjadi [TIM-03]. Skema topologi *star* dapat dilihat dalam Gambar 2.3.



Gambar 2.3 Topologi *Star*

Sumber: [FCI-05]

Keuntungan :

Paling fleksibel karena pemasangan kabel mudah, penambahan atau pengurangan kabel mudah dan tidak mengganggu bagian jaringan lain, kontrol terpusat akan memudahkan dalam deteksi dan isolasi kesalahan/kerusakan, dan akan memudahkan dalam pengelolaan jaringan.

Kerugian :

Boros kabel, kontrol terpusat jadi elemen kritis.

2.1.1.4 *Tree*

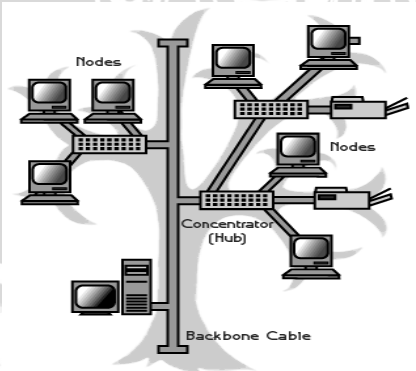
Topologi *tree* merupakan gabungan dari topologi *linear bus* dan *star*. Topologi *tree* terdiri dari beberapa grup topologi *star* di mana tiap grup tersebut terkoneksi ke sebuah kabel *backbone –linear bus*- [FCI-05]. Skema topologi *tree* dapat dilihat dalam Gambar 2.4.

Keuntungan:

Mudah dikembangkan.

Kerugian :

Jika kabel *backbone* bermasalah, maka antar segmen tidak dapat saling berhubungan.



Gambar 2.4 Topologi *Tree*
Sumber: [FCI-05]

2.1.1.5 *Mesh*

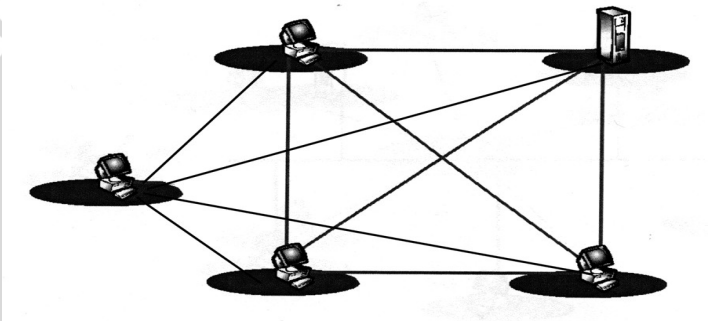
Pada topologi *mesh* setiap terminal memiliki kabel yang terhubung ke terminal lain [FCI-05]. Skema topologi *mesh* dapat dilihat dalam Gambar 2.5.

Keuntungan :

Kerusakan satu kabel tidak akan merusak komunikasi diantara dua komputer.

Kerugian :

Biaya yang dibutuhkan sangat mahal, dan instalasi yang sulit. Dibutuhkan lebih banyak kabel di mana tiap terminal harus memiliki kabel sendiri ke semua terminal yang ada.



Gambar 2.5 Topologi *Mesh*

Sumber: [MIL-05]

2.1.2 *Metropolitan Area Network* (MAN)

MAN merupakan versi LAN yang berukuran lebih besar dan memakai teknologi yang sama dengan LAN [TIM-03]. MAN merupakan pilihan tepat untuk membangun jaringan komputer antar kantor dalam satu kota. MAN dapat mencakup perusahaan yang memiliki kantor–kantor yang letaknya sangat berdekatan dan MAN mampu menunjang data dan suara, bahkan dapat disambungkan dengan jaringan televisi kabel. Jaringan ini memiliki jarak dan radius 10-50 km [TIM-03].

2.1.3 *Wide Area Network* (WAN)

Wide Area Network adalah sebuah jaringan yang memiliki jarak yang sangat luas, karena radiusnya mencakup sebuah negara dan benua [TAN-03]. Pada sebagian

besar WAN, komponen yang dipakai dalam berkomunikasi terdiri dari dua komponen yaitu: kabel transmisi dan elemen *switching*. Kabel transmisi berfungsi untuk memindahkan bit-bit dari satu komputer ke komputer lainnya, sedangkan elemen *switching* di sini adalah sebuah komputer khusus yang digunakan untuk menghubungkan dua buah kabel transmisi atau lebih. Saat data tiba dari sebuah jalur masukan, elemen *switching* kemudian menggunakan algoritma *routing* yang dimilikinya untuk memilih jalur keluaran mana data tersebut harus diteruskan. Komputer *switching* ini disebut *router* [TAN-03].

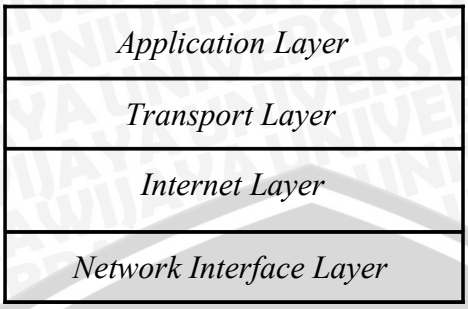
2.2 *Transmission Control Protocol/Internet Protocol* (TCP/IP)

Protokol merupakan sejumlah aturan yang mengatur format dan arti sebuah *frame*, paket atau pesan yang dipertukarkan di antara dua *peer entity* dalam sebuah *layer* (lapisan) [TAN-04]. *Entity* adalah elemen – elemen aktif pada sebuah *layer*. Setiap *entity* dapat berupa *software entity* (seperti halnya sebuah proses), atau *hardware entity* (misalnya *network interface card* (NIC) *chip*) [TAN-04]. *Entity–entity* yang ada pada *layer* yang sama namun pada mesin yang berbeda disebut *peer enti*ty. Protokol *TCP/IP* adalah kumpulan dari protokol yang memungkinkan komunikasi pada bermacam–macam jaringan [LEO-04]. Protokol TCP/IP memungkinkan banyak komputer dari semua ukuran, dan dari banyak vendor komputer yang berbeda, dengan sistem operasi yang berbeda pula untuk saling berkomunikasi [STE-94].

TCP/IP terdiri dari sekumpulan protokol yang masing-masing bertanggungjawab atas bagian-bagian tertentu dari komunikasi data [LEO-04]. TCP/IP terdiri dari empat lapis kumpulan protokol yang bertingkat. Keempat lapis/*layer* tersebut dari puncak adalah :

- *Application Layer*
- *Transport Layer*
- *Internet Layer*
- *Network Interface Layer*

Susunan lapisan protokol TCP/IP dapat dilihat dalam Gambar 2.6.



Gambar 2.6 *Layer* TCP/IP

Sumber: [LEO-04]

Pada TCP/IP data akan mengalami proses enkapsulasi data dari protokol yang berada pada satu *layer* ke protokol yang berada pada *layer* lain [PUR-98]. Data yang telah dienkapsulasi akan dikirim melalui jalur fisik ke alamat tujuan [PUR-98].

2.2.1 *Application Layer*

Application layer adalah *layer* yang mengatur perincian dari aplikasi utama [STE-94]. Pada *layer* ini terdapat bermacam – macam protokol tingkat tinggi, antara lain:

- TELNET (*virtual terminal protokol*)

  TELNET adalah protokol yang digunakan untuk melakukan *remote* ke mesin/ komputer lain [PUR-98].

- FTP (*File Transfer Protocol*)

  FTP adalah protokol yang digunakan untuk memindahkan data secara efektif dari satu mesin ke mesin lain [PUR-98].

- SMTP (*Simple Mail Transport Protocol*)

  SMTP adalah protokol yang digunakan untuk mengirim *e-mail* dengan cara yang mudah dan cepat [PUR-98].

- HTTP (*Hypertext Transfer Protocol*)

HTTP adalah protokol yang digunakan untuk layanan akses situs pada jaringan TCP/IP [PUR-98].

- DNS (*Domain Name Service*)

  DNS adalah protokol yang digunakan untuk memetakan nama-nama *host* ke nomor Internet Protokol(IP) dan dari nomor IP ke nama *host* [PUR-98].

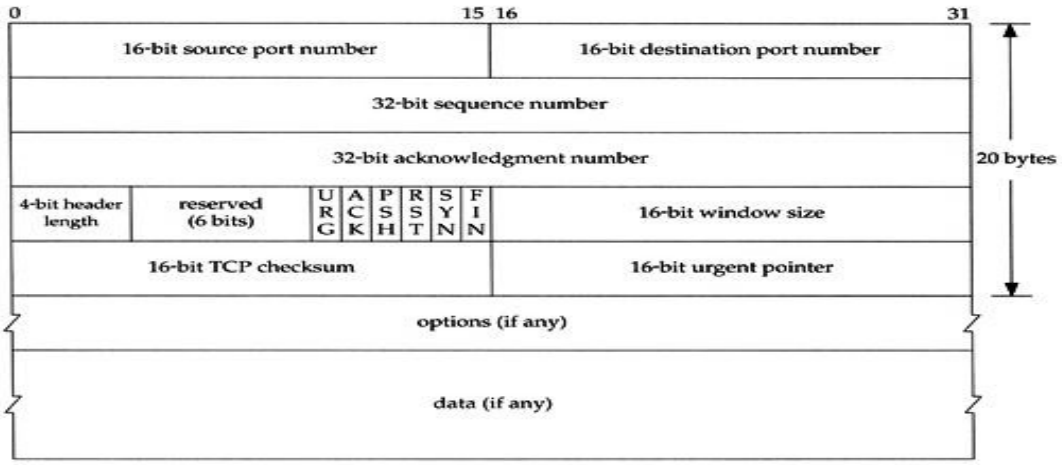
2.2.2 *Transport Layer*

Transport Layer adalah *layer* yang dirancang agar *peer entity – peer entity* pada *host* sumber dan *host* tujuan memungkinkan melakukan komunikasi [TAN-04]. *Transport layer* merupakan *layer* komunikasi data yang mengatur aliran data antara dua *host*, untuk keperluan aplikasi *layer* di atasnya [TAN-04]. Ada dua buah protokol pada *layer* ini, yaitu TCP dan UDP.

2.2.2.1 *TCP* (*Transmission Control Protocol*)

TCP merupakan protokol yang berorientasi pada hubungan yang andal (*reliable connection-oriented protocol*) yang mengijinkan sebuah aliran *byte* yang berasal pada suatu mesin untuk dikirimkan tanpa *error* ke sebuah mesin yang ada di internet[17] [TAN-04]. Protokol TCP dikenal sebagai : *connection oriented*, *reliable*, dan *byte stream service* [PUR-98]. *Connection oriented* berarti sebelum melakukan pertukaran data, dua aplikasi pengguna TCP harus melakukan pembentukan hubungan (*handshake*) terlebih dahulu. *Reliable* berarti TCP menerapkan proses deteksi kesalahan paket dan retransmisi. *Byte stream service* berarti paket dikirimkan ke tujuan secara berurutan. Format Segmen *TCP* ditunjukkan dalam Gambar 2.7.

[17] internet (dengan "i" kecil) adalah rangkaian komputer yang saling berhubungan membentuk suatu jaringan komputer yang besar.



Gambar 2.7 Format Segmen TCP

Sumber: [STE-94]

Source port dan *destination port* :

field ini berisi angka yang mengidentifikasi aplikasi pengirim dan penerima segmen TCP.

Sequence number :

berisi nomor urut *byte stream* dalam data aplikasi yang dikirim.

acknowledgment number :

field ini mengidentifikasikan *sequence number* berikutnya yang diharapkan oleh penerima pada setiap kali data ini sukses dikirim jika ACK bit diset 1.

Header length :

berisi panjang *header* TCP. Dengan lebar 4 bit, field ini harus merepresentasikan panjang *header* TCP dalam satuan 4 byte. Jika 4 bit ini berisi 1 (1111 biner = 15 desimal), maka panjang *header* maksimal adalah 15 x 4 = 60 byte.

Reserved :

field ini dicadangkan untuk penggunaan di masa yang akan datang dan harus di set 0.

URG :

jika bit ini di-set ke 1 maka *Urgent Pointer* sedang digunakan.

ACK :

bit ini diset 1 untuk mengindikasikan bahwa *acknowledge number* adalah validbila ACK sama dengan 0, segmen tidak mengandung *acknowledgement* sehingga *acknowledge number* akan diabaikan*PSH*: mengindikasikan data yang di-*push.* Penerima dalam hal ini diminta untuk mengantarkan data ke aplikasi ketika itu sampai dan tidak mem-buffer-kannya sampai sepenuhnya telah diterima.

RST :

digunakan untuk menyetel ulang koneksi yang telah menjadi kacau sehubungan telah terjadinya *crash* dan sebab-sebab lainnya. Bit ini juga dipakai untuk membuang segmen yang tidak valid.

SYN :

digunakan untuk membentuk koneksi, permintaan koneksi memiliki SYN=1 dan ACK=0 untuk menandakan field *acknowledgement* tidak dipakai. Pada dasarnya bit SYN digunakan untuk menandakan CONNECTION REQUEST (indikasi *host* meminta koneksi, SYN=1 dan ACK=0) dan CONNECTION ACCEPTED (indikasi *host* menerima koneksi, SYN=1 dan ACK=1).

FIN :

digunakan untuk melepaskan koneksi. Bit ini menspesifikasikan bahwa pengirim tidak mempunyai data lainnya yang akan ditransmisikan.

Window size :

Ukuran window dari pengirim yang akan diterima dalam format oktet, merupakan banyak byte maksimal yang bisa diterima setiap saat. Lebar *field* ini adalah 16 bit (2 byte), sehingga nilai maksimalnya adalah 65535.

Cheksum :

berisi angka hasil perhitungan matematis yang digunakan untuk memeriksa kesalahan data.

Urgent Pointer :

field diangggap sah apabila URG di set 1.

Options :

field Option digunakan untuk memberikan fungsi lain yang tidak terdapat pada *header* biasa.

Data :

Data upper layer

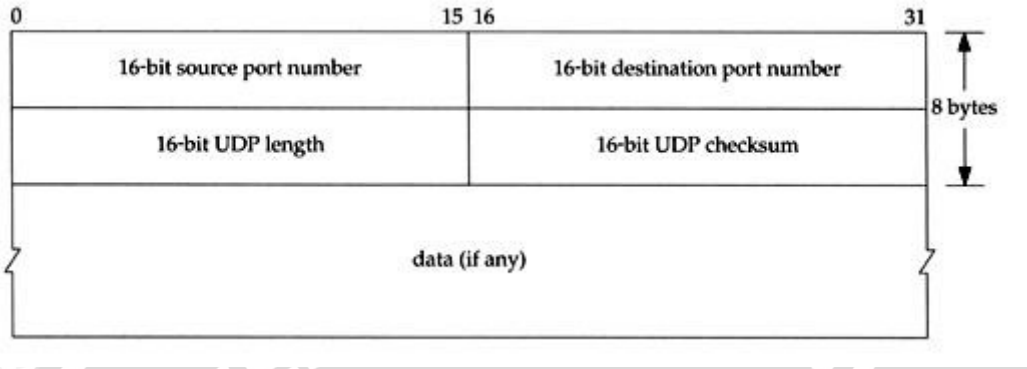
2.2.2.2 *UDP* (*User Datagram Protocol*)

UDP merupakan protokol yang tidak andal (*unreliable*) dan tanpa sambungan (*connectionless*) bagi aplikasi–aplikasi yang tidak memerlukan pengurutan TCP atau pengendalian aliran dan bagi aplikasi–aplikasi yang ingin melayani dirinya sendiri. [TAN-04]. Perbedaan TCP dengan UDP adalah TCP bersifat *connection oriented* dan UDP bersifat *connectionless*. Pada UDP tidak ada *sequencing* (pengurutan kembali) paket yang datang, *acknowledgment* terhadap paket yang datang, atau retransmisi jika paket mengalami masalah di tengah jalan. Kemiripan UDP dengan TCP ada pada penggunaan *port number*. UDP menggunakan *port number* ini untuk membedakan pengiriman *datagram* ke beberapa aplikasi berbeda yang terletak pada komputer yang sama [PUR-98].

UDP umumnya dipakai untuk *transfer* data yang memerlukan kecepatan tetapi kurang peka terhadap kesalahan, seperti *transfer* suara dan video [PUR-98]. UDP ini bersifat *broadcasting*[18] atau *multicasting*[19]. *Broadcasting* dan *multicasting* hanya dapat diterapkan pada UDP, di mana memungkinkan suatu aplikasi mengirimkan pesan

[18] *broadcast* adalah transmisi paket data di mana data dikirim ke semua perangkat didalam jaringan.
[19] *multicast* adalah transmisi paket data di mana data dikirim ke beberapa perangkat didalam jaringan.

tunggal pada banyak penerima [STE-94]. Pengiriman datagram ke banyak klien sekaligus akan efisien jika prosesnya menggunakan metode *connectionless*. Dalam Gambar 2.8 ditunjukkan format dari datagram UDP.



Gambar 2.8 Format Datagram UDP

Sumber: [STE-94]

Source port dan *destination port* :

field ini berisi angka yang mengidentifikasi aplikasi pengirim dan penerima segmen UDP.

UDP length :

berisi panjang *datagram–header* dan data UDP

C*hecksum* :

berisi angka hasil perhitungan matematis yang digunakan untuk memeriksa kesalahan data.

Data :

Data upper layer.

2.2.3 *Internet Layer*

Internet layer adalah *layer* yang dirancang agar memungkinkan *host* mengirimkan paket ke jaringan dan memungkinkan paket-paket itu berjalan sendiri-sendiri ke tujuannya (yang besar kemungkinan berada di jaringan lain) [TAN-04].

Paket-paket ini mungkin tiba di tujuan dengan urutan yang berbeda dengan urutan saat dikirimkan. Dalam kasus seperti ini, *layer-layer* yang berada di atasnya bertugas untuk mengatur kembali, bila pengiriman terurut diinginkan. Protokol yang ada pada *layer* ini antara lain : IP, ICMP, dan ARP.

2.2.3.1 IP (*Internet Protocol*)

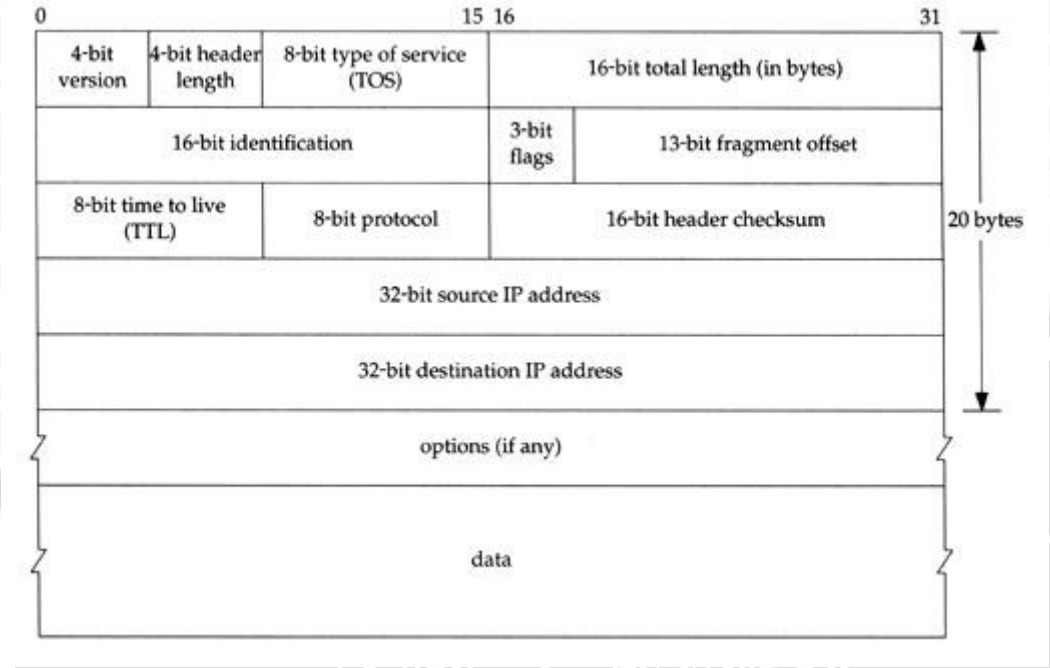
IP yang akan dibahas pada sub bab ini adalah *Internet Protocol version 4* (IPv4). Protokol IP merupakan inti dari protokol TCP/IP [PUR-98]. Seluruh data yang berasal dari protokol pada *layer* di atas IP diolah oleh protokol IP, dan dipancarkan sebagai datagram IP agar sampai ke tujuan. IP memiliki sifat yang dikenal sebagai *unreliable, connectionless*, dan *datagram delivery service* [PUR-98].

Unreliable berarti bahwa protokol IP tidak menjamin bahwa paket yang dikirim pasti sampai ke tujuan [PUR-98]. Protokol IP hanya mampu melakukan usaha sebaik-baiknya (*best effort delivery service*), agar paket yang dikirim tersebut sampai ke tujuan. Jika di perjalanan paket tersebut terjadi hal-hal yang tidak diinginkan (salah satu jalur putus, *router* mengalami kongesti, atau *host*/*network* tujuan sedang *down*), maka protokol IP hanya memberitahukan pengirim paket melalui protokol ICMP, bahwa terjadi masalah dalam pengiriman paket IP ke tujuan. Jika diinginkan keandalan yang lebih baik, keandalan itu harus disediakan oleh protokol yang berada di atas *layer* IP ini (yaitu TCP dan aplikasi pengguna) [PUR-98].

Connectionless berarti dalam mengirim paket dari tempat asal ke tujuan, pihak pengirim dan penerima paket IP sama sekali tidak mengadakan perjanjian (*handshake*) terlebih dahulu [PUR-98].

Datagram delivery service berarti setiap paket data yang dikirim adalah *independen* terhadap paket data yang lain [PUR-98]. Hal ini menyebabkan jalur yang ditempuh oleh masing-masing paket data IP ke tujuannya bisa berbeda satu dengan lainnya. Kedatangan paket bisa tidak berurutan karena jalur yang ditempuh berbeda [PUR-98].

Metode tersebut dipakai dalam pengiriman paket IP untuk menjamin tetap sampainya paket IP ke tujuan, walaupun salah satu jalur ke tujuan itu mengalami masalah.



Gambar 2.9 Format Datagram IP
Sumber: [STE-94]

Dalam Gambar 2.9 diberikan format datagram IPv4. Setiap paket IPv4 membawa data yang terdiri dari :

- *Version* (4 bit), berisi versi dari protokol IP yang dipakai, yaitu versi 4.
- *Header Length* (4 bit), berisi panjang dari *header* paket IP ini dalam hitungan 32 bit word.
- *Type of Service* (8 bit), berisi kualitas *service* yang dapat mempengaruhi cara penanganan paket IP ini.
- *Total Length of Datagram* (16 bit), berisi panjang IP *datagram* total dalam ukuran byte.

- *Identification* (16 bit), diperlukan untuk mengijinkan *host* tujuan menentukan *datagram* pemilik *fragment* yang baru datang. Semua *fragment* suatu *datagram* berisi nilai *identification* yang sama.

- *Flag* (3 bit), 1 buah *field* 1-bit tidak dipakai, 2 buah *field* 1-bit yang dipakai merupakan DF (*Don't Fragment*) dan MF (*More Fragment*) . Bit DF (*Don't Fragment*) merupakan perintah ke *router* agar tidak melakukan *fragmentasi* terhadap datagram karena mesin tujuan tidak memiliki kemampuan untuk menggabungkan kembali potongan-potongan. Bit MF (*More Fragment*) diperlukan untuk mengetahui apakah semua *fragment datagram* tiba. Seluruh *fragment* kecuali yang terakhir memiliki bit ini.

- *Fragment Offset* (13 bit), memberitahukan diantara *datagram* mana yang ada pada saat itu yang memiliki *fragment* yang bersangkutan. Seluruh *fragment* kecuali yang terakhir, karena tersedia 13 bit, maka terdapat nilai maksimum 8192 *fragment* per *datagram*, yang menghasilkan panjang *datagram* maksimum 65.536 byte, di mana lebih besar dari *field total length*

- *Time to Live* (8 bit) adalah *counter* yang digunakan untuk membatasi umur paket. *Field* ini diharapkan dapat menghitung waktu dalam detik, yang memungkinkan umur maksimum 255 detik. *Counter* harus diturunkan pada setiap *hop*[20] dan diturunkan beberapa kali bila berada dalam antrian *router* dalam waktu yang lama. Pada kenyataannya *field* ini hanya menghitung *hop*. ketika *counter* mencapai nol, paket dibuang dan paket peringatan dikirim kembali ke *host* sumber. Fitur ini mencegah *datagram* agar terus menerus berjalan, sesuatu yang mungkin terjadi bila tabel *routing* rusak.

- *Protocol* (8 bit), mengandung angka yang mengidentifikasikan protokol *layer* atas pengguna isi data dari paket IP ini.

- *Header Checksum* (16 bit), berisi nilai *checksum* yang dihitung dari seluruh *field* dari *header* paket IP. Sebelum dikirimkan, protokol IP terlebih dahulu menghitung *checksum* dari *header* paket IP tersebut untuk nantinya dihitung

[20] *hop* adalah satuan langkah, dari satu router ke router berikutnya, yang dilewati paket data untuk menuju alamat tujuan.

kembali di sisi penerima. Jika terjadi perbedaan, maka paket ini dianggap rusak dan dibuang.

- IP address pengirim dan penerima data (masing-masing 32 bit), berisi alamat pengirim paket dan penerima paket.

- Beberapa byte *option*, diantaranya :

  Strict Source Route. Berisi daftar lengkap IP *address* dari *router* yang harus dilalui oleh paket ini dalam perjalanannya ke *host* tujuan. Selain itu paket balasan atas paket ini, yang mengalir dari *host* tujuan ke *host* pengirim, diharuskan melalui *router* yang sama.

  Loose Source Route. Dengan mengeset *option* ini, paket yang dikirim diharuskan singgah di beberapa *router* seperti yang disebutkan dalam *field option* ini. Jika diantara kedua *router* yang disebutkan terdapat *router* lain, paket masih diperbolehkan melalui *router* tersebut.

2.2.3.2 ICMP (*Internet Control Message Protocol*)

ICMP adalah protokol yang bertugas mengirimkan pesan-pesan kesalahan dan kondisi lain yang memerlukan perhatian khusus [PUR-98]. Pesan/paket ICMP dikirim jika terjadi masalah pada *layer* IP dan *layer* atasnya (TCP/UDP) [PUR-98].

Ada dua tipe pesan yang dapat dihasilkan oleh ICMP yaitu *ICMP Error Message* dan *ICMP Query Message* [PUR-98]. *ICMP Error Message* dihasilkan jika terjadi kesalahan pada jaringan. Sedangkan *ICMP Query Message* adalah jenis pesan yang dihasilkan oleh protokol ICMP jika pengirim paket menginginkan informasi tertentu yang berkaitan dengan kondisi jaringan [PUR-98].

2.2.3.3 ARP (*Address Resolution Protocol*)

Protokol ARP (*Address Resolution Protocol*) adalah protokol yang digunakan untuk melakukan pemetaan IP *address* ke *ethernet address*. Paket IP umumnya dikirim melalui *card ethernet* pada jaringan lokal [PUR-98]. *Ethernet address* digunakan untuk komunikasi *card ethernet* satu dengan lainnya *Ethernet address* ini besarnya 48 bit.

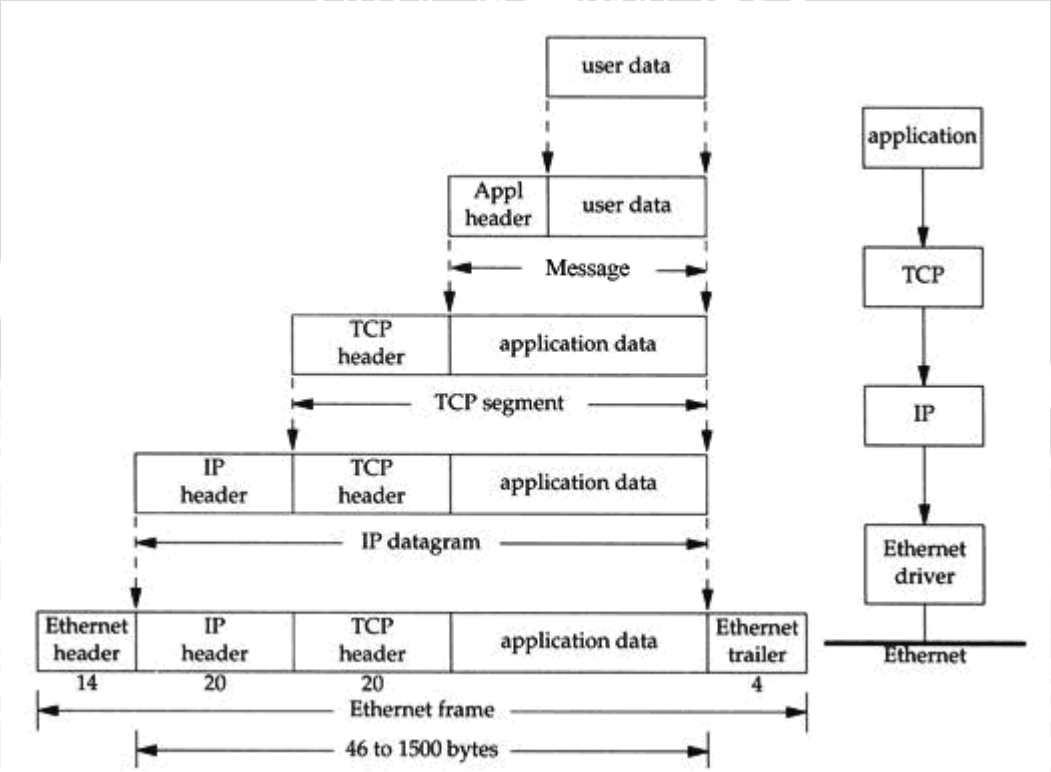
Setiap card *ethernet* memiliki *ethernet address* yang berbeda-beda [PUR-98]. Pada saat hendak mengirimkan data ke komputer dengan IP tertentu, suatu *host* pada jaringan *ethernet* perlu mengetahui, pada *ethernet address* yang manakah tempat IP tersebut terletak [PUR-98].

2.2.4 *Network Interface Layer*

Layer ini adalah *layer* yang dirancang untuk mengirim dan menerima data dari media fisik. Beberapa contohnya ialah *ethernet*, SLIP dan PPP [PUR-98].

2.2.5 Enkapsulasi Data

Enkapsulasi data adalah proses pembentukan dan pembungkusan data dalam format paket tertentu untuk setiap *layer* TCP/IP [PUR-98]. Data dari *application layer* yang akan dikirimkan melalui jaringan akan mengalami proses enkapsulasi data pada setiap *layer* di bawahnya. Proses ini ditunjukkan dalam Gambar 2.10.



Gambar 2.10 Proses *Data Encapsulation*

Sumber: [STE-94]

Pada setiap lapisan yang dilalui ditambahkan sebuah *header* yang berisi informasi-informasi pengontrol komunikasi. Unit data yang akan diterima pada *transport layer* dibagi-bagi menjadi potongan data yang disebut segmen TCP atau *datagram* UDP untuk aplikasi yang menggunakan UDP sebagai protokol *transport*-nya. Dalam Gambar 2.10 diberikan contoh untuk potongan data segmen TCP. Segmen tersebut diteruskan ke *internet layer* dan disebut datagram IP. Datagram IP diteruskan pada *network interface layer* untuk ditransmisikan melalui media fisik jaringan [PUR-98].

2.3 *Quality of Service*

Quality of Service (QoS) adalah suatu pengukuran tentang seberapa baik jaringan dan merupakan suatu usaha untuk mendefinisikan karakteristik dan sifat dari suatu servis [FER-98]. QoS biasanya digunakan untuk mengukur sekumpulan atribut performansi yang telah dispesifikasikan dan biasanya diasosiasikan dengan suatu servis. Pada jaringan berbasis IP, IP QoS mengacu pada performansi dari paket-paket IP yang lewat melalui satu atau lebih jaringan.

QoS didesain untuk membantu *end user* menjadi lebih produktif dengan memastikan bahwa *user* mendapatkan performansi yang handal dari aplikasi-aplikasi berbasis jaringan [FER-98].

QoS mengacu pada kemampuan jaringan untuk menyediakan layanan yang lebih baik pada trafik jaringan tertentu melalui teknologi yang berbeda-beda. QoS merupakan suatu tantangan yang cukup besar dalam jaringan berbasis IP dan internet secara keseluruhan. Tujuan dari QoS adalah untuk memuaskan kebutuhan-kebutuhan layanan yang berbeda, yang menggunakan infrastruktur yang sama. QoS menawarkan kemampuan untuk mendefinisikan atribut-atribut layanan jaringan yang disediakan, baik secara kualitatif maupun kuantitatif [FER-98].

Banyak hal dapat terjadi pada paket saat paket dikirimkan dari pengirim ke penerima. Ini menimbulkan masalah-masalah yang dapat meimbulkan pernurunan performa jaringan. Masalah-masalah tersebut antara lain :

- **Pen-*drop*-an packet**

  Sebuah perangkat jaringan mungkin gagal mengirim paket jika paket tersebut tiba ketika *buffer*-nya sudah penuh. Beberapa, tidak sama sekali, atau semua paket mungkin akan di *drop* tergantung dari kondisi dari jaringan, dan tidak mungkin untuk mengetahui apa yang akan terjadi. Aplikasi penerima dapat meminta paket ini untuk dikirimkan ulang, di mana dapat menyebabkan *delay* pada keseluruhan pengiriman.

- ***Delay***

  Mungkin akan membutuh waktu yang panjang untuk paket tiba ditujuannya, karena paket tertahan di antrian yang panjang, atau mengambil jalur yang memutar untuk menghindari kongesti. Kemudian, paket mungkin akan mengambil jalur langsung yang lebih cepat. Akhirnya *delay* sangat sulit di prediksi.

- ***Jitter***

  Paket dari pengirim akan tiba di penerima dengan *delay* yang berbeda. Variasi dari *delay* dikenal dengan nama *jitter* dan dapat berpengaruh dalam kualitas dari streamming audio dan/atau video.

- **Pengiriman *Out-of-order***

  Ketika sekumpulan paket yang saling berhubungan dikirimkan melalui Internet, paket yang berbeda dapat mengambil jalur yang berbeda, masing-masing menghasilkan *delay* yang berbeda. Hasilnya adalaha paket tiba dengan urutan yang berbeda dengan ketika dikirimkan. Masalah ini menyebabkan dibutuhkannya protokol tambahan khusus yang berfungsi untuk menyusun kembali paket yang tidak terurut menjadi terurut saat tiba di penerima. Hal ini sangat penting untuk video dan stream VoIP di mana kualitas sangat dipengaruhi oleh latency dan urutan paket.

- ***Error***

  Terkadang paket mengalami salah kirim, rusak, atau tergabung dengan paket lain, ketika dalam pengiriman. Penerima harus mendeteksi ini dan, sama seperti jika paket di-*drop*, meminta pengirim mengirim ulang paket tersebut.

Komponen-komponen dari QoS adalah:

- ***Reliability***, merupakan kehandalan jalur yang dilalui suatu paket dari pengirim ke penerima melalui jaringan.

- ***Delay***, merupakan total waktu yang dilalui suatu paket dari pengirim ke penerima melalui jaringan. *Delay* dari pengirim ke penerima pada dasarnya tersusun atas hardware latency, *delay* akses, dan *delay* transmisi. *Delay* yang paling sering dialami oleh trafik yang lewat adalah *delay* transmisi, yang dapat dirumuskan sebagai berikut:

$$Delay = \frac{packet\_size \times 8}{line\_speed} \quad (2.1)$$

  Keterangan :

  Delay = besar delay, dalam sekon.

  packet_size = ukuran paket data, dalam *byte*

  line_speed = kecepatan *link*, dalam *bit* per sekon.

  Untuk aplikasi-aplikasi suara dan video interaktif, kemunculan dari *delay* akan mengakibatkan sistem seperti tak merespon.

- ***Jitter***, merupakan variasi dari *delay* end-to-end. Level-level yang tinggi pada *jitter* dalam aplikasi-aplikasi berbasis UDP merupakan situasi yang tidak dapat diterima di mana aplikasi-aplikasinya merupakan aplikasi-aplikasi real-time, seperti sinyal audio dan video. Pada kasus seperti itu, *jitter* akan menyebabkan sinyal terdistorsi, yang dapat diperbaiki hanya dengan meningkatkan buffer di antrian. *Jitter* dapat dirumuskan sebagai berikut:

$$jitter = jitter + \frac{|D(i-1,i)| - jitter}{16} \quad (2.2)$$

dengan

$$D(i,j)=(Rj-Ri)-(Sj-Si)=(Rj-Sj)-(Ri-Si) \quad (2.3)$$

Keterangan:

jitter = besar *jitter*, dalam sekon

D = beda waktu yang timbul antara paket data ke i dengan paket data ke i - 1.

R = waktu paket diterima

S = waktu paket dikirim

i = paket data yang diambil sebagai sampel

j = paket data ke i - 1

- ***Bandwidth***, merupakan *rate* transfer data maksimal yang dapat diteruskan antara dua titik.

Bersama-sama semua komponen ini menentukan QoS yang dibutuhkan oleh tiap-tiap *flow*. Beberapa aplikasi yang umum dan kebutuhan QoS tiap-tiap aplikasi ditunjukkan di dalam Gambar 2.11.

Application	Reliability	Delay	Jitter	Bandwidth
E-mail	High	Low	Low	Low
File transfer	High	Low	Low	Medium
Web access	High	Medium	Low	Medium
Remote login	High	Medium	Medium	Low
Audio on demand	Low	Low	High	Medium
Video on demand	Low	Low	High	High
Telephony	Low	High	High	Low
Videoconferencing	Low	High	High	High

Gambar 2.11 Kebutuhan QoS untuk tiap-tiap aplikasi

Sumber: [ALM-01]

Empat aplikasi teratas dalam gambar 2.11 memiliki kebutuhan yang tinggi terhadap *realibility*. Tidak boleh ada *bit* yang terkirim salah. Ini dicapai dengan melakukan *checksum* pada setiap paket yang diterima. Jika sebuah paket rusak, akan langsung di buang dan pengiriman ulang akan dilakukan. Empat aplikasi berikutnya memiliki toleransi yang lebih tinggi terhadap *error*, jadi *checksum* tidak dilakukan.

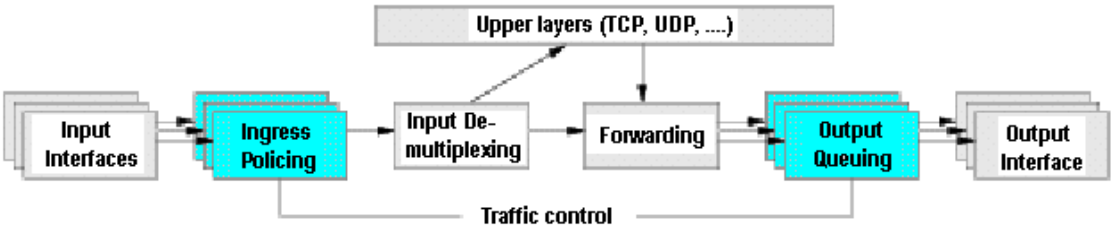
Aplikasi pengiriman file, termasuk *e-mail* dan video, tidak sensitif terhadap *delay*. Jika semua paket mengalami *delay* selama beberapa detik, tidak timbul masalah. Aplikasi yang sifatnya interaktif, seperti *web surfing* dan *remote login*, lebih sensitif terhadap *delay*. Aplikasi *real-time*, seperti telepon dan konferensi video memiliki kebutuhan yang ketat terhadap *delay*. Jika semua kata dalam panggilan telepon mengalami *delay* sebesar 2 detik, maka *user* akan merasa bahwa koneksi tidak dapat di terima. Di sisi lain, memainkan file audio dan video dari sebuah server tidak memerlukan *delay* yang rendah.

Tiga aplikasi teratas tidak sensitif terhadap paket yang tiba dalam waktu yang tidak seragam. *Remote login* sedikit sensitif terhadap hal itu, karena karakter di layar akan terlihat mengalami *burst* jika *jitter* sangat besar. Video dan audio sangat sensitif terhadap *jitter*. Jika *user* melihat sebuah video melewati sebuah jaringan dan semua *frame* mengalami *delay* sebesar 2 detik, maka tidak ada masalah yang timbul. Tapi jika waktu transmisi mengalami variasi secara acak, hasilnya akan sangat buruk. Pada audio, *jitter* walau hanya beberapa millisekon akan dapat terdengar.

Tiap-tiap aplikasi juga berbeda dalam kebutuhan *bandwidth*, dengan *e-mail* dan *remote login* tidak memerlukan *bandwidth* yang besar, tapi video dalam segala bentuknya memerlukan *bandwidth* yang besar.

2.4 *Linux Traffic Control*

Prinsip dasar dari *Linux Traffic Control* dapat dijelaskan dalam gambar 2.12.



Gambar 2.12 *Linux Traffic Control*

Sumber: [ALM-01]

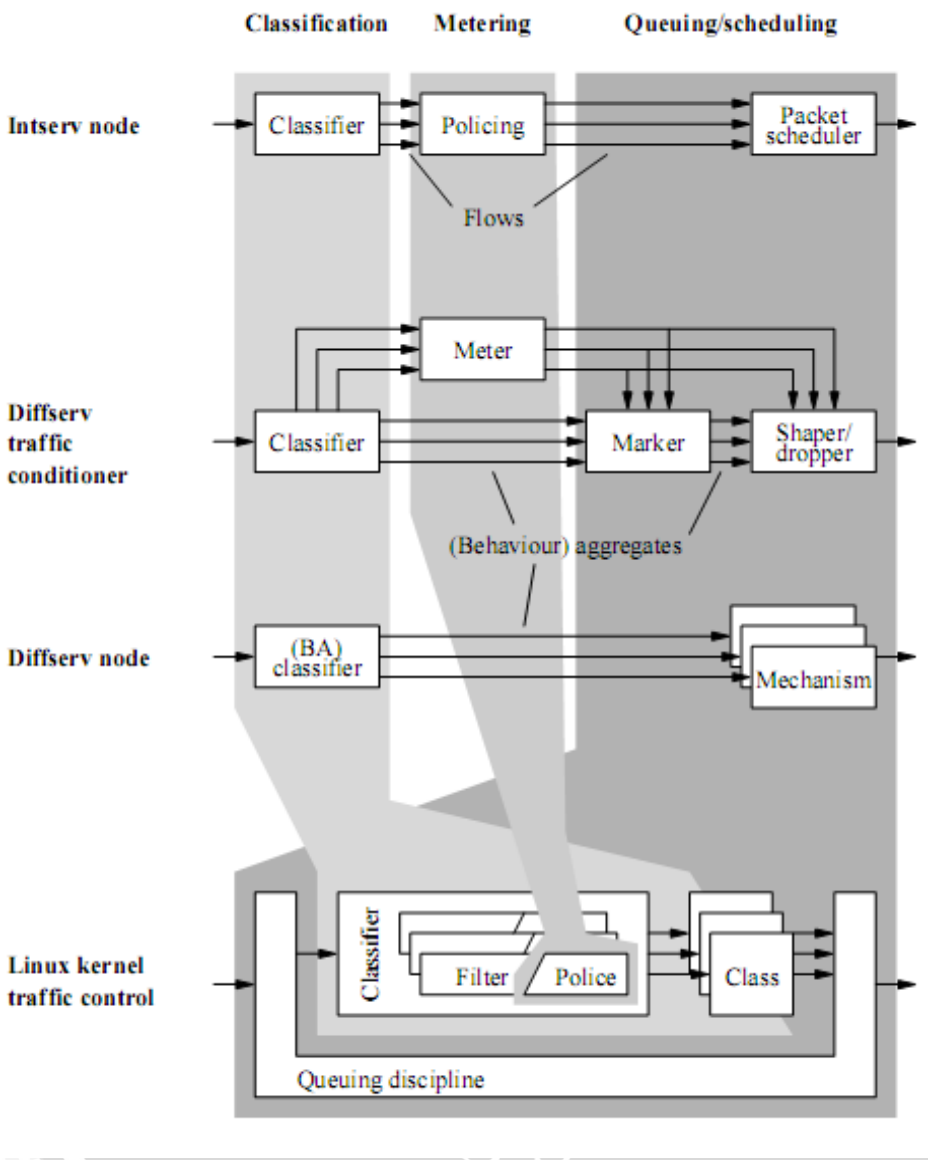
Gambar 2.12 menunjukkan bagaimana kernel Linux memproses paket yang datang. Paket datang melalui *Input interface,* di mana paket tersebut diatur oleh *Ingress policing*. *Ingress policing* akan membuang paket yang tidak diinginkan, misal trafik datang terlalu cepat. Setelah melalui *Ingress policing paket* diteruskan ke *Input de-multiplexer. Input de-multiplexer* akan memeriksa apakah paket yang datang ditujukan untuk *node* lokal. Jika tidak, maka paket akan diteruskan ke blok *forwarding*. Jika ya, maka paket akan dikirimkan ke *layer* yang lebih tinggi untuk pemrosesan lebih lanjut. *Layer* yang lebih tinggi mungkin juga menghasilkan paket dan menyerahkannya ke *layer* yang lebih rendah untuk tugas-tugas seperti enkapsulasi, *routing*, dan pada akhirnya pengiriman.

Blok *forwarding* berfungsi antara lain untuk pemilihan *output interface*, pemilihan *hop* berikutnya, dan enkapsulasi. Setelah itu, paket tersebut akan diantrikan untuk ditransmisikan pada *output interface*. Ini adalah titik kedua fungsi dari *traffic control* akan diterapkan. *Traffic control* dapat digunakan antara lain untuk menentukan apakah paket diantrikan atau di buang, dalam urutan seperti apa paket dikirimkan, dan ia juga dapat menahan pengiriman suatu paket [ALM-01].

Traffic control dapat digunakan untuk membangun kombinasi yang kompleks dari *queuing disciplines* (disiplin antrian), *classes* (kelas-kelas), dan *filter* yang akan mengontrol paket-paket yang dikirimkan pada *interface output*.

Mekanisme *Linux traffic control* memberikan dasar landasan dukungan untuk mengembangkan *integrated services* (*intserv*) dan *differentiated services* (*diffserv*) di

Linux. Gambar 2.13 menunjukkan model arsitektur dan terminologi yang digunakan oleh IETF dan bagaimana relasinya dengan elemen *Linux traffic control*.



Gambar 2.13 Landasan untuk pengembangan "intserv" dan "deffserv"

Sumber: [ALM-01]

Sesuai dengan yang ditunjukkan dalam gambar 2.13, manajemen *bandwidth* di Linux terdiri dari tiga blok pembangun dasar, yaitu:

1. *Queuing disciplines*
2. *Classes* (didalam q*ueuing disciplines*)

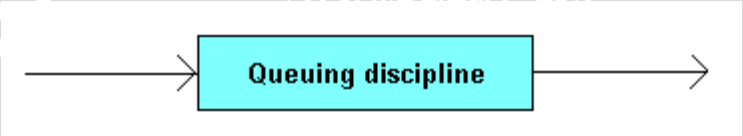
3. *Filters*

4. *Policing*

2.5 *Queuing Disciplines*

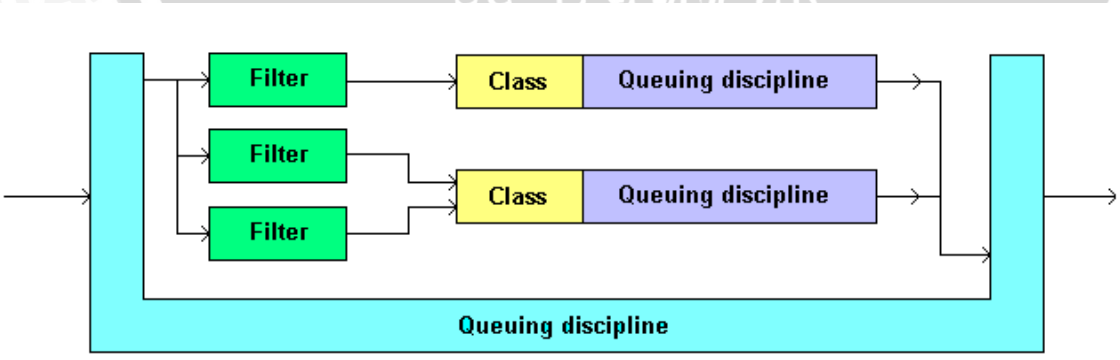
Queuing disciplines (disiplin antrian) adalah algoritma yang menentukan bagaimana antrian paket akan dikirimkan. Setiap perangkat jaringan dapat diasosiasikan dengan suatu disiplin antrian [ALM-01].

Sebuah disiplin antrian sederhana dapat berupa sebuah antrian, di mana semua paket disimpan dalam urutan bagaimana mereka diantrikan, dan dikosongkan secepat perangkat tersebut mengirimkannya. Contoh disiplin antrian tersebut ditunjukkan dalam Gambar 2.14 di mana strukur internal tidak terlihat dari luar.



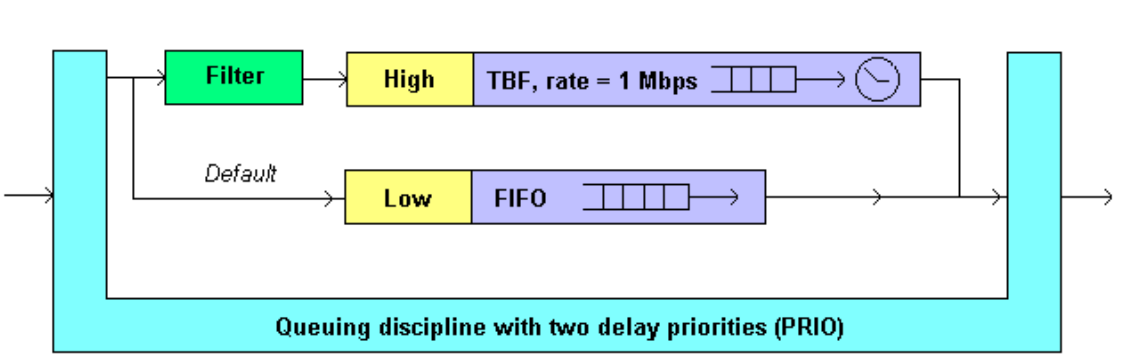
Gambar 2.14 Disiplin antrian sederhana tanpa kelas
Sumber: [ALM-01]

Disiplin antrian yang lebih rumit mungkin menggunakan *filter* untuk membedakan antara kelas paket yang berbeda dan memproses tiap kelas dengan cara yang khusus [ALM-01]. Contoh disiplin antrian tersebut ditunjukkan dalam Gambar 2.15. Catat bahwa lebih dari satu *filter* dapat dipetakan pada kelas yang sama.



Gambar 2.15 Disiplin antrian sederhana dengan kelas lebih dari satu
Sumber: [ALM-01]

Disiplin antrian dan kelas saling terkait satu sama lain. Keberadaan kelas dan sematiknya merupakan atribut penting dari disiplin antrian. Kontras dengan itu, *filter* dapat digabungkan disiplin antrian dan kelas, selama disiplin antrian tersebut memiliki kelas. Kelas umumnya tidak menyimpan paket mereka sendiri, tapi mereka menggunakan disiplin antrian lain untuk melakukannya. Disiplin antrian tersebut dapat di pilih dari beberapa disiplin antrian yang sudah ada, dan mungkin juga memliki kelas, yang pada gilirannya menggunakan disiplin antrian, dan seterusnya [ALM-01]. Contoh disiplin antrian tersebut ditunjukkan dalam Gambar 2.16.



Gambar 2.16 Kombinasi dari disiplin antrian prioritas, *Token Bucket Filter* (TBF), dan *First In First Out* (FIFO)

Sumber: [ALM-01]

Disiplin antrian diidentifikasi menggunakan sebuah nomor sepanjang 32 *bit*, yang dibagi menjadi nomor mayor dan nomor minor (masing-masing 16 *bit*). Format penulisannya adalah *mayor*:*minor*. Untuk disiplin antrian, nomor minor selalu nol. Percobaan untuk membuat disiplin antrian dengan nomor minor bukan nol akan menghasilkan *error* [ALM-01].

Nomor mayor harus unik untuk tiap *interface*, contoh : hanya boleh ada satu disiplin antrian 1:0 untuk *eth0*, tapi mungkin ada disiplin antrian lain 1:0 untuk *eth1*.

Nomor mayor yang diberikan oleh *user* harus dalam batas 1 sampai dengan 32767. Nomor mayor 0 berarti ”tidak didefinisikan”, dan nomor mayor 32768 sampai dengan 65535 otomatis dialokasikan oleh kernel jika membuat disiplin antrian dengan nomor mayor yang tidak didefinisikan.

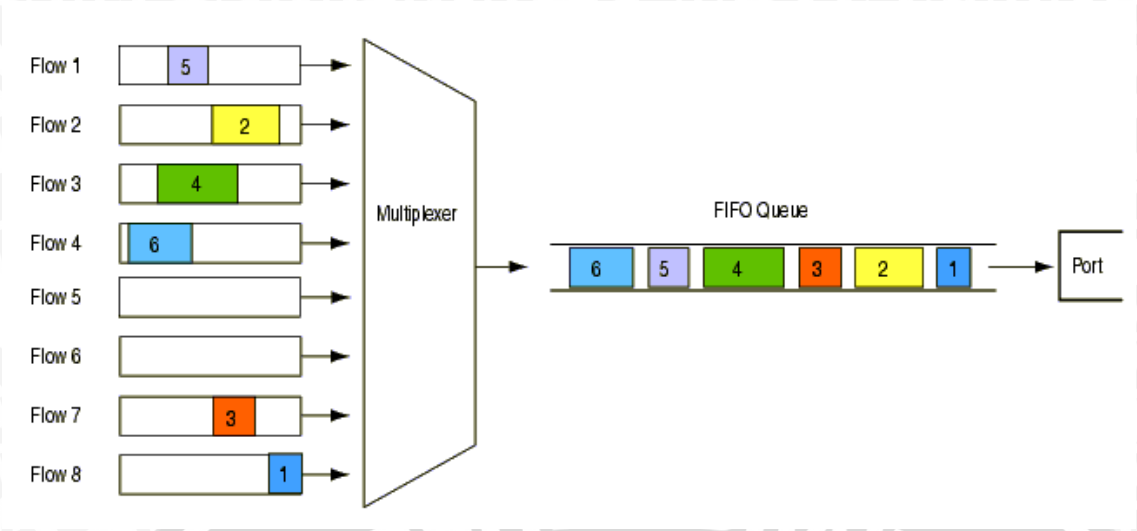
Dalam Linux, dikenal 2 macam disiplin antrian, yaitu *classless queuing discipline* (disiplin antrian tanpa kelas) dan *classful queuing discipline* (disiplin antrian dengan kelas).

2.5.1 *Classless Queuing Discipline*

Classless queuing discipline merupakan suatu disiplin antrian yang hanya dapat menunda ataupun men-*drop* (membuang) paket yang diterimanya. Ini dapat digunakan untuk men-*shape* (mengatur) trafik pada semua *interface output* tanpa adanya pembagian-pembagian kelas lagi. *Classless queueing discipline* yang ada di Linux antara lain : FIFO, TBF, dan SFQ.

8.1.1 **FIFO *Queueing***

FIFO queueing merupakan teknik antrian yang paling sederhana dan umum digunakan. Antrian ini dapat bekerja dengan baik apabila *link-link* tidak mengalami kongesti. Paket pertama yang ditempatkan pada antrian di *interface output* adalah paket yang akan pertama kali meninggalkan *interface* tersebut. Masalah yang akan muncul pada antrian ini adalah jika sebuah *node* melakukan transfer file, ia akan memakai semua *bandwidth* pada sebuah *link* yang akan merugikan suatu sesi yang interaktif. Fenomena ini mengacu pada barisan paket yang disebabkan satu sumber mengirimkan suatu "baris" paket ke tujuannya, sehingga paket dari *node* lain terjebak di belakang barisan paket tersebut. Antrian ini sangat efektif untuk *link-link* yang besar yang memiliki *delay* yang kecil dan kongesti yang minimal [SEM-01].

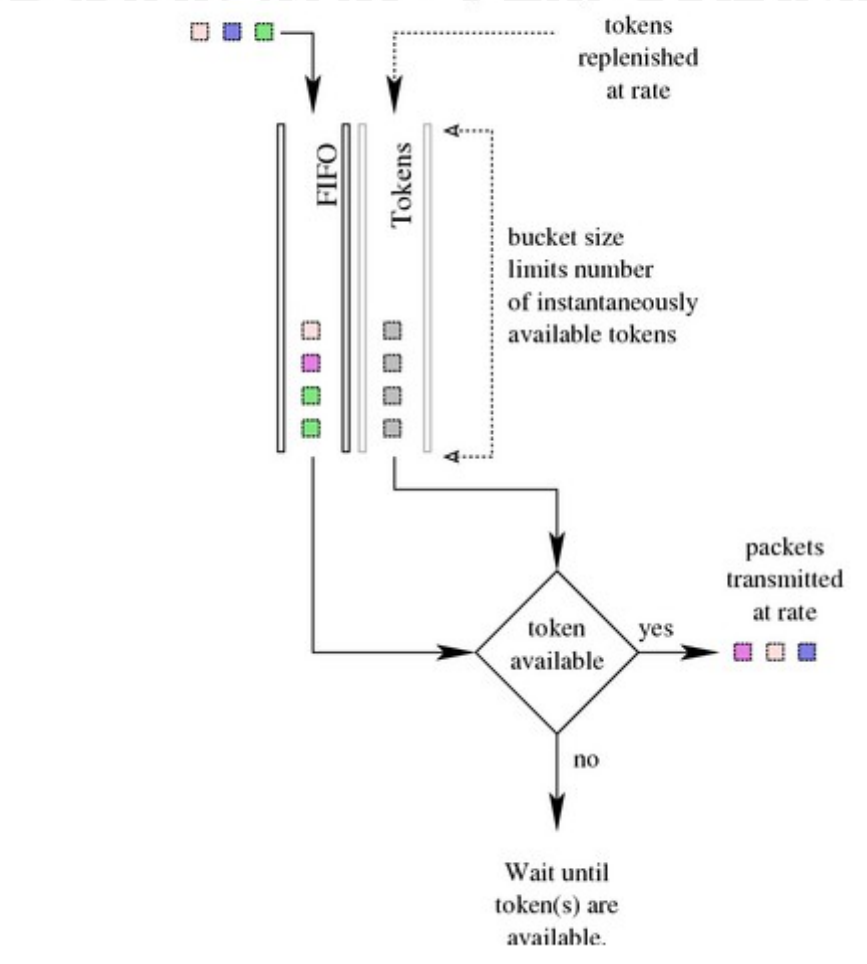


Gambar 2.17 Disiplin antrian FIFO

Sumber: [SEM-01]

2.5.1.2 *Token Bucket Filter* (TBF)

Token bucket merupakan suatu definisi formal dari suatu transfer *rate*. Antrian ini memiliki tiga buah komponen: ukuran *burst*, *mean rate*, dan interval waktu (Tc) [CIS-04].



Gambar 2.18 Disiplin antrian TBF

Sumber: [BRO-06]

Implementasi TBF terdiri dari sebuah buffer (bucket), yang secara konstan diisi oleh beberapa informasi virtual yang dinamakan token, pada *rate* yang spesifik (token *rate*). Parameter paling penting dari bucket adalah ukurannya, yaitu banyaknya token yang dapat disimpan [HUB-02].

Setiap token yang masuk mengumpulkan satu paket yang datang dari antrian data dan kemudian dihapus dari bucket. Dengan menghubungkan algoritma ini dengan dua aliran – token dan data, akan kita dapati tiga buah kemungkinan skenario:

- Data yang datang pada TBF memiliki *rate* yang sama dengan masuknya token. Dalam hal ini, setiap paket yang masuk memiliki tokennya masing-masing dan akan melewati antrian tanpa adanya *delay*.

- Data yang datang pada TBF memiliki *rate* yang lebih kecil daripada *rate* token. Hanya sebagian dari token yang dihapus pada output pada tiap paket data yang dikirim ke antrian, sehingga token akan menumpuk, memenuhi ukuran bucket. Token yang tidak digunakan kemudian akan dapat digunakan untuk mengirimkan data pada kecepatan yang melampaui *rate* token standar, ini terjadi jika ada ledakan data yang pendek.

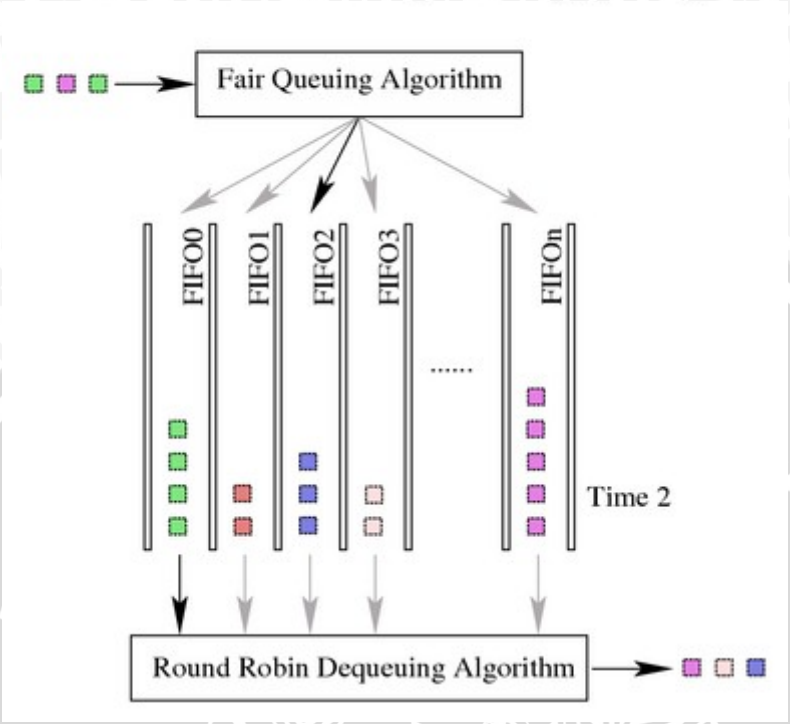
- Data yang datang pada TBF memiliki *rate* yang lebih besar daripada *rate* token. Hal ini berarti bucket akan segera kosong dari token, yang akan menyebabkan TBF akan menutup alirannya untuk sementara. Hal inilah yang dinamakan situasi overlimit. Jika paket-paket tetap datang, maka paket-paket akan segera di *drop*.

Skenario terakhir ini sangatlah penting, karena dengan skenario ini kita dapat menshaping *bandwidth* yang tersedia terhadap data yang melewati filter.

2.5.1.3 *Stochastic Fairness Queueing* (SFQ)

Stochastic Fairness Queueing (SFQ) merupakan suatu implementasi sederhana dari keluarga algoritma *fair queueing*. Ia kurang akurat apabila dibandingkan dengan yang lain, tetapi ia juga membutuhkan perhitungan yang lebih sedikit dibandingkan dengan yang lain. Kata kunci dari SFQ adalah *conversation* (atau aliran), yang sangat berhubungan dengan sebuah sesi TCP atau sebuah aliran UDP. Trafik akan dibagi ke dalam beberapa jumlah besar antrian FIFO, satu untuk setiap *conversation*. Trafik kemudian dikirim secara *round-robin*, dengan memberikan kesempatan kepada tiap sesi untuk mengirimkan data pada gilirannya. SFQ disebut *stochastic* karena ia sebenarnya tidak menyediakan sebuah antrian untuk setiap sesi, ia memiliki suatu algoritma yang membagi trafik melalui sejumlah antrian yang terbatas dengan menggunakan algoritma *hashing*. Karena *hash* inilah, sesi yang banyak dapat berakhir di bucket yang sama, yang akan membagi dua tiap sesi kesempatan untuk mengirimkan paket, sehingga membagi dua kecepatan efektif yang tersedia. Untuk menghindari situasi ini menjadi terlihat, SFQ mengubah algoritma hashing yang ia miliki secara sering sehingga dua sesi yang bertabrakan akan terjadi dalam waktu yang singkat. SFQ hanya berguna jika

interface outgoing yang kita miliki benar-benar penuh. Jika tidak, maka tidak akan ada queue pada mesin linux kita sehingga tidak akan ada efeknya [CIS-04].



Gambar 2.19 Disiplin antrian SFQ
Sumber: [BRO-06]

2.5.2 *Classful Queueing Discipline*

Classful Queueing Discipline merupakan suatu disiplin antrian yang akan membagi trafik berdasarkan kelas-kelas. *Classful queueing discipline* sangat berguna apabila kita memiliki trafik yang berbeda-beda yang harus memiliki pembedaan penanganan. Ketika trafik memasuki suatu *class*ful queueing discipline, maka paket tersebut akan dikirimkan ke kelas-kelas di dalam *queueing discipline*, dengan kata lain paket tersebut perlu diklasifikasikan terlebih dahulu. Untuk menentukan apa yang harus dilakukan dengan sebuah paket yang datang, maka filter-filter akan digunakan. Harus dipahami bahwa filter-filter ini dipanggil dari dalam sebuah *queueing discipline*, bukan sebaliknya. Filter-filter yang terdapat pada qdisc tersebut akan menghasilkan suatu keputusan, dan *queueing discipline* akan menggunakan hasil ini untuk mengantrikan paket ke salah satu kelas yang telah tersedia. Setiap subklas mungkin saja akan mencoba filter-filter yang lainnya untuk melihat apakah ada instruksi lain yang berlaku. Jika

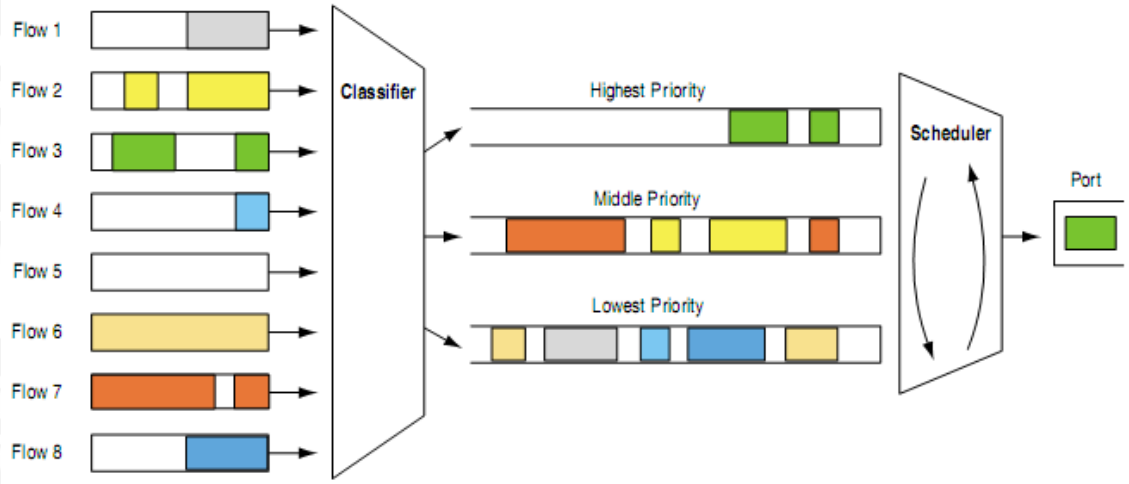
tidak, maka kelas akan mengantrikan paket tersebut ke *queueing discipline* yang ia miliki [HUB-02].

Di samping memiliki suatu *queueing discipline*, kebanyakan dari *classful queueing discipline* juga menerapkan fungsi *shaping* (mengatur). Ini sangat berguna untuk melakukan penjadwalan paket (misal dengan SFQ) dan pengontrolan *rate* sekaligus. Kita akan sangat membutuhkan proses ini apabila kita memiliki suatu *interface* dengan kecepatan tinggi (misal ethernet) ke suatu alat yang lebih lambat (misal modem). *Classfull queueing discipline* yang ada di Linux antara lain : PRIO, CBQ, dan HTB.

8.2.1 PRIO

Priority queueing memungkinkan manajer jaringan untuk menentukan bagaimana trafik diprioritaskan dalam suatu jaringan. Dengan menentukan serangkaian filter berdasarkan karakteristik dari paket, trafik ditempatkan pada suatu antrian; antrian dengan prioritas yang lebih tinggi dilayani lebih dahulu, kemudian antrian yang lebih rendah lagi dilayani sesuai urutannya. Jika antrian dengan prioritas paling tinggi selalu penuh, maka antrian ini akan secara kontinyu dilayani dan paket-paket dari antrian yang lain akan di *drop*. Dalam algoritma antrian ini, maka satu jenis trafik jaringan dapat mendominasi trafik-trafik lainnya. *Priority queueing* akan menentukan trafik ke dalam salah satu dari 4 antrian: tinggi, sedang, normal, dan rendah [CIS-04].

Priority Queueing dapat menjamin bahwa trafik-trafik yang penting mendapatkan penanganan yang tercepat pada setiap titik di mana ia digunakan. *Queueing* ini di buat untuk memberikan prioritas yang ketat pada trafik-trafik yang penting. *Priority queueing* dapat secara fleksibel membagi prioritas berdasarkan protokol jaringan (misal IP, IPX, atau AppleTalk), *interface* yang masuk, ukuran paket, alamat asal/tujuan, dan sebagainya [CIS-04].



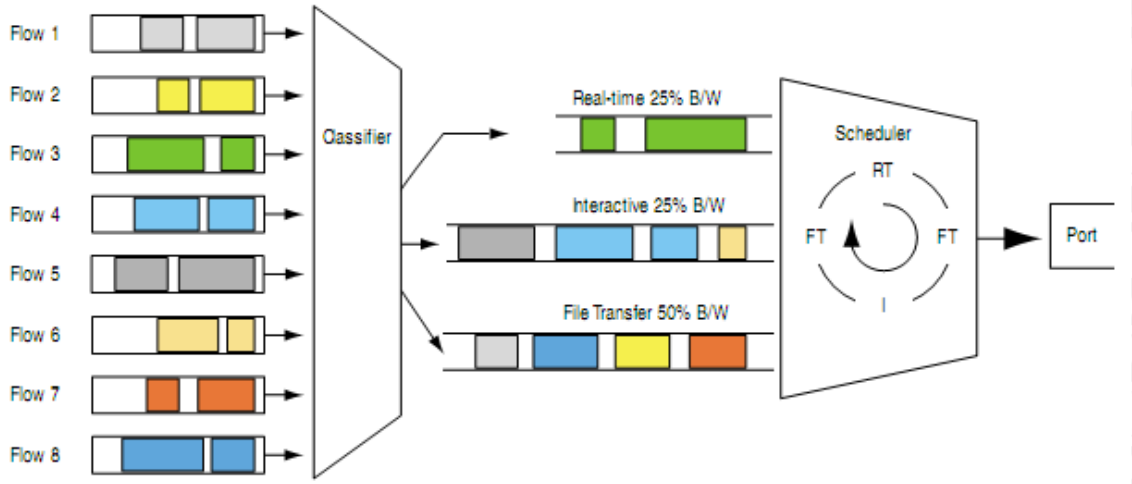
Gambar 2.20 Disiplin antrian PRIO

Sumber: [SEM-01]

8.2.2 *Class Based Queueing* (CBQ)

CBQ dapat menerapkan pembagian kelas dan menshare *link bandwidth* melalui struktur kelas-kelas secara hirarki. Setiap kelas memiliki antriannya masing-masing dan diberikan jatah *bandwidth*-nya. Sebuah kelas child dapat meminjam *bandwidth* dari kelas parent selama terdapat kelebihan *bandwidth* [HUB-02]. Gambar di bawah ini menunjukkan komponen dasar dari CBQ. CBQ bekerja sebagai berikut: *class*ifier akan mengarahkan paket-paket yang datang ke kelas-kelas yang bersesuaian. Estimator akan mengestimasi *bandwidth* yang sedang digunakan oleh sebuah kelas. Jika sebuah kelas telah melampaui limit yang telah ditentukannya, maka estimator akan menandai kelas tersebut sebagai kelas yang *overlimi*t. *Scheduler* menentukan paket selanjutnya yang akan dikirim dari kelas-kelas yang berbeda-beda, berdasarkan pada prioritas dan keadaan dari kelas-kelas. *Round robin scheduling*[21] digunakan antara kelas-kelas dengan prioritas yang sama. *Class Based Queueing* disebut juga *Weighted Round Robin* atau *Custom Queueing* [SEM-01].

[21] *Round robin scheduling* adalah suatu metode penjadwalan dimana setiap entitas yang menggunakan suatu sumber daya diberikan waktu yang sama untuk menggunakan sumber daya tersebut dan menggunakan sumber daya tersebut secara bergiliran



Gambar 2.21 Disiplin antrian CBQ

Sumber: [SEM-01]

Parameter *Class Based Queuing* (CBQ) :

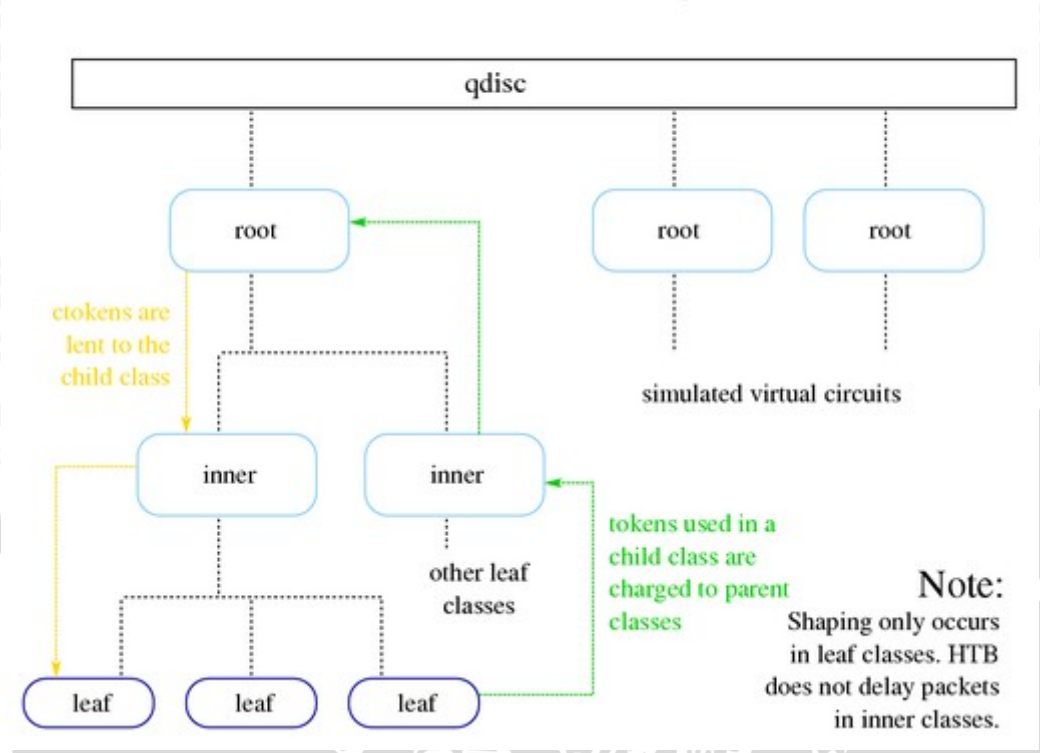
- *Root*, menandakan *link* utama (*main link*) dalam suatu hirarki *link sharing*.
- *Handle,* berperan dalam menentukan suatu penanganan unik oleh pengguna terhadap *queuing discipline*.
- *Bandwidth*, merupakan *bandwidth* maksimum yang tersedia bagi suatu kelas
- *Parent,* digunakan untuk menangani *queuing discipline* pada sebuah hirarki atau merupakan identitas kelas yang akan dipakai pada saat membangun hirarki kelas-kelas dibawahnya.
- *Avpkt,* merupakan besar paket rata-rata pada sebuah kelas
- *Allot,* merupakan MTU dan MAC Header
- *Classid*, merepresentasikan penamaan suatu kelas yang diberikan oleh pengguna
- *Rate*, merupakan representasi besarnya *bandwidth* yang dialokasikan pada sebuah kelas
- *Cell*, mewakili batas byte pada paket-paket yang ditransmisikan

- *Weight*, berguna untuk proses Weighted Round Robin (WRR) yaitu memberikan kesempatan yang proporsional dalam mengirimkan paket berdasarkan pada alokasi *bandwidth*nya
- *Maxburst*, merupakan parameter untuk jumlah byte yang akan dikirim pada saat terjadi letupan/*burst* terpanjang yang mungkin.
- *Minburst*, merepresentasikan jumlah byte yang akan dikirim pada kondisi letupan terpendek yang mungkin.
- *Bounded*, menandakan suatu kelas tidak dapat meminjam *bandwidth* yang tak terpakai pada kelas-kelas diatasnya.
- *Isolated*, mengindikasikan suatu kelas tidak akan meminjamkan *bandwidth* yang tersisa pada kelas lain.
- *Split*, umumnya terdapat pada root CBQ. Split bisa diset pada node-node hirarki sehingga akan mengaktifkan penggunaan pengklasifikasi kelas yang mudah dan cepat.
- *Defmap*, bila ada paket yang tidak sesuai dengan pengklasifikasi kelas, maka akan disediakan prioritas secara logik.
- *Prio*, merepresentasikan prioritas yang diberikan suatu kelas.
- *Protocol*, digunakan oleh filter untuk mengidentifikasi paket-paket yang dimiliki oleh protokol tersebut.
- *Limit*, parameter panjang maksimal timbunan antrian.

8.2.3 ***Hierarchical Token Bucket*** **(HTB)**

HTB merupakan salah satu disiplin antrian yang memiliki tujuan untuk menerapkan *link sharing* secara presisi dan adil. Dalam konsep *link sharing*, jika suatu kelas meminta kurang dari jumlah service yang telah ditetapkan untuknya, sisa

bandwidth akan didistribusikan ke kelas-kelas yang lain yang meminta service [DEN-03].



Gambar 2.22 Konsep *Link Sharing* pada HTB

Sumber: [BRO-06]

Parameter *Hierarchical Token Bucket* (HTB) :

- *Classid*, merepresentasikan penanganan suatu kelas yang diberikan oleh pengguna.
- *Rate*, merupakan representasi besarnya *bandwidth* yang dialokasikan pada sebuah kelas
- *Ceil*, merupakan *bandwidth* maksimum yang dapat dipakai bagi suatu kelas.
- *Burst*, merupakan jumlah data yang dapat dikirim pada kecepatan maksimum dari suatu kelas.

HTB menggunakan TBF sebagai estimator yang sangat mudah diimplementasikan. Estimator ini hanya menggunakan parameter *rate*, sebagai

akibatnya seseorang hanya perlu mengeset *rate* yang akan diberikan ke suatu kelas. Oleh karena itu HTB lebih mudah dan intuitif dibandingkan CBQ [DEN-03].

Pada HTB terdapat parameter *ceil* sehingga kelas akan selalu mendapatkan *bandwidth* di antara base *rate* dan nilai *ceil rate*nya. Parameter ini dapat dianggap sebagai estimator kedua, sehingga setiap kelas dapat meminjam *bandwidth* selama *bandwidth* total yang diperoleh memiliki nilai di bawah nilai *ceil*. Hal ini mudah diimplementasikan dengan cara tidak mengijinkan proses peminjaman *bandwidth* pada saat kelas telah melampaui *rate* ini (keduanya *leaf* dan interior dapat memiliki *ceil*). Sebagai catatan, apabila nilai *ceil* sama dengan nilai base *rate*, maka akan memiliki fungsi yang sama seperti parameter *bounded* pada CBQ, di mana kelas-kelas tidak diijinkan untuk meminjam *bandwidth*. Sedangkan jika nilai *ceil* diset tak terbatas atau dengan nilai yang lebih tinggi seperti kecepatan *link* yang dimiliki, maka akan didapat fungsi yang sama seperti kelas *non-bounded*. Sebagai contoh, seseorang dapat menjamin *bandwidth* 1 Mbit untuk suatu kelas, dan mengijinkan penggunaan *bandwidth* sampai dengan 2 Mbit pada kelas tersebut apabila *link* dalam keadaan *idle* [DEN-03].

Untuk menjadwalkan pengiriman paket dari antrian, maka HTB menggunakan suatu proses penjadwalan yang dapat dijelaskan sebagai berikut:

Keterangan:

- ***Class*** merupakan parameter yang diasosiasikan dengan *rate* yang dijamin (*assured rate*) AR, *ceil rate* CR, prioritas P, level dan quantum. *Class* dapat memiliki parent. Selain AR dan CR, didefinisikan juga actual *rate* atau R, yaitu *rate* dari aliran paket yang meninggalkan *class* dan diukur pada suatu perioda waktu tertentu.

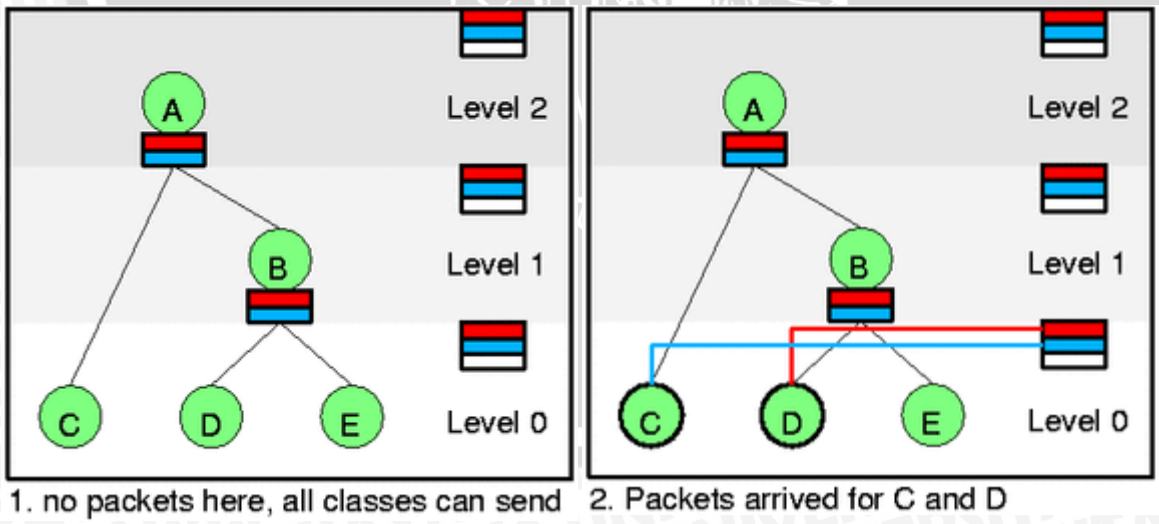
- ***Leaf*** merupakan *class* yang tidak memiliki anak. Hanya *leaf* yang dapat memegang antrian paket.

- **Level** dari kelas menentukan posisi dalam suatu hirarki. *Leaf-leaf* memiliki level 0, root *class* memiliki level=jumlah level-1 dan setiap *inner class* memiliki level kurang dari satu dari parentnya. Untuk gambar di bawah (jumlah level=3).

- **Mode** dari *class* merupakan nilai-nilai buatan yang diperhitungkan dari R, AR, dan CR. Mode-mode yang mungkin adalah:
    - **Merah**: **R** > **CR**
    - **Kuning**: **R** <= **CR** and **R** > **AR**
    - **Hijau** selain yang di atas

Diasumsikan terdapat kelas-kelas yang tersusun seperti pohon pada HTB *scheduler*. Dalam tiap level, terdapat sebuah *self feed* yang terdiri dari *self slot*. Sehingga jika misalkan terdapat tiga buah level, maka terdapat 6 buah *self slot* (diabaikan dulu *slot* berwarna putih). Setiap *self slot*, memegang daftar kelas-kelas, daftar tersebut digambarkan melalui garis berwarna yang terhubung dari *self slot* menuju kelas. *Self slot* berisikan daftar dari semua kelas-kelas berwarna hijau yang memiliki permintaan.

Setiap kelas *inner* (non-*leaf*), memiliki *inner feed slot*. *Inner feed* memiliki sebuah *inner feed slot* per prioritas (berwarna merah untuk prioritas tinggi, berwarna biru untuk prioritas rendah) dan per *inner class*. *Inner slot* akan menangani kelas-kelas anak yang berwarna kuning.



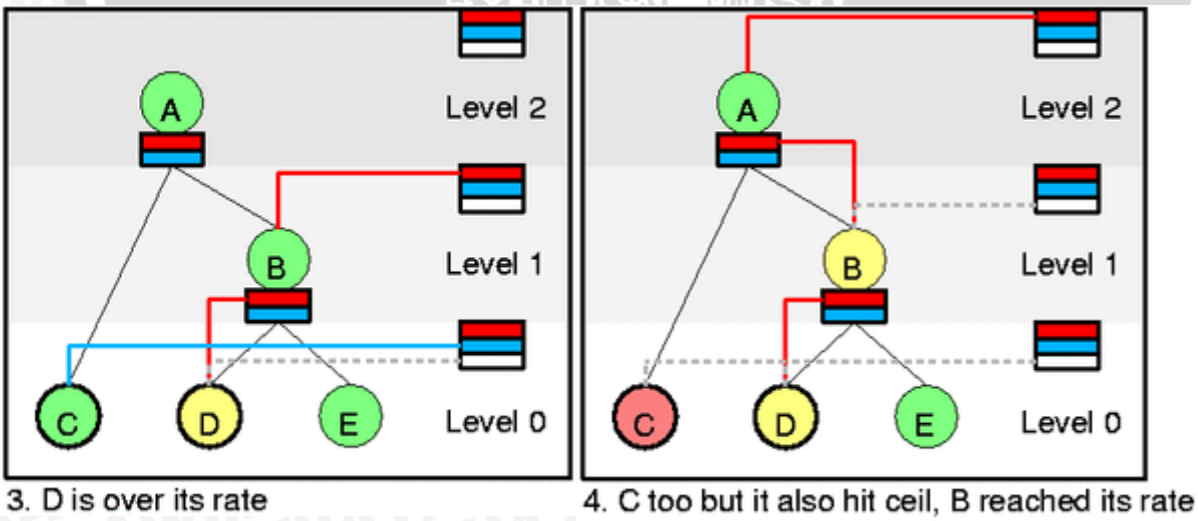
Gambar 2.23 Cara Kerja *Scheduler* di HTB

Sumber: [DEN-03]

Slot berwarna putih yang terdapat pada gambar di atas berfungsi sebagai antrian tunggu per level. Ia akan memegang daftar kelas-kelas yang berada pada levelnya yang berwarna merah atau kuning.

Dari sini dapat terlihat apabila dapat di jaga semua kelas pada daftar yang bersesuaian, maka proses pemilihan paket selanjutnya yang akan melakukan pengiriman paket dari antrian akan sangatlah mudah. HTB cukup melihat pada daftar *self feed* dan memilih *slot* yang tidak kosong (yaitu *slot* yang terhubung dengan garis ke sebuah kelas) dengan level yang paling rendah dan prioritas yang paling tinggi. Dalam gambar 2.23 (1) di atas tidak terdapat *slot* yang tidak kosong, sehingga tidak ada paket yang akan dikirim. Dalam gambar 2.23 (2), terdapat *slot* berwarna merah dengan level 0 pada kelas D, sehingga kelas D dapat mengirim paketnya.

Untuk lebih jelasnya dapat dijelaskan sebagai berikut. Dalam gambar 2.23 (1), tidak terdapat *leaf* yang memiliki tumpukan antrian (semua *leaf* yang memiliki tumpukan antrian digambarkan dengan lingkaran tipis) sehingga tidak ada yang dapat dilakukan. Pada gambar 2.23 (2), paket-paket datang untuk C dan D. Sehingga HTB akan mengaktifkan kelas-kelas ini, dan karena keduanya berwarna hijau, HTB akan menambahkannya ke *self slot* yang bersesuaian. Proses *dequeue* akan memilih D karena D berada pada level yang rendah dan memiliki prioritas yang lebih tinggi daripada C.



Gambar 2.24 Cara Kerja *Scheduler* di HTB

Sumber: [DEN-03]

Diasumsikan paket telah dikirim dari kelas D dan HTB mengukur berapa besar paket yang dikirim melalui bantuan TBF. Karena paket yang dikirim melampaui besar *rate* yang telah ditetapkan untuk kelas D, maka HTB akan memaksa D untuk berubah warna menjadi kuning. Sebagai bagian dari perubahan ini, HTB akan menghapus D dari daftar self *feed* dan menambahkan D ke *inner feed* B. HTB juga secara rekursif akan menambahkan B ke self *feed* pada levelnya. Karena D akan kembali ke keadaan hijau lagi pada suatu waktu, maka HTB akan menambahkan D ke *slot* putih pada level 0.

Proses dequeue sekarang akan memilih kelas C meskipun kelas C memiliki prioritas yang lebih rendah dibandingkan D. Ini dikarenakan C berada pada daftar *slot* dengan level yang lebih rendah. Secara intuitif pun hal ini baik dilakukan, untuk apa meminjam *bandwidth* jika ada yang dapat mengirim paket tanpa meminjam *bandwidth*.

- TBF lebih akurat dalam menentukan *rate*.
- TBF tidak memerlukan waktu akhir pengiriman yang tepat seperti pada ewma-*idle*. Akibatnya, HTB akan bekerja secara kuat pada semua perangkat jaringan di mana CBQ menunjukkan *rate* yang aneh dan salah.
- HTB memungkinkan *cross-device bandwidth sharing*. Sehingga ethernet pertama dapat memakai *bandwidth* dari ethernet kedua.

8.2 *Classes*

Class (kelas) adalah penggolongan trafik berdasarkan *network*, alamat IP *address*, dan/atau nomor *port*.

Seperti yang telah disebutkan, disiplin antrian dan kelas saling berhubungan. Setiap kelas memiliki antrian sendiri, yang secara default merupakan antrian FIFO. Ketika sebuah fungsi pengantrian dari suatu disiplin antrian dipanggil, disiplin antrian menggunakan filter untuk menentukan kelas mana paket tersebut seharusnya berada. Kemudian fungsi pengantrian dari disiplin atrian untuk kelas ini dipanggil [RAD-99].

Ada dua cara di mana sebuah kelas dapat dikenali. Satu, dengan menggunakan tanda pengenal kelas, yang ditentukan oleh pengguna. Tanda pengenal yang lain, yang

digunakan kernel untuk mengenali sebuah kelas, dikenal sebagai tanda pengenal internal. Tanda pengenal ini unik dan diberikan oleh disiplin antrian [RAD-99].

8.3 *Filters*

Filter digunakan untuk mengklasifikasikan paket data berdasarkan sifat-sifat dari paket data, contohnya: TOS di header IP, alamat IP, nomor port, dan lain-lain. Filter berjalan jika fungsi pengantrian dari disiplin pengantrian berjalan. Disiplin antrian menggunakan filter untuk menentukan letak kelas dari tiap-tiap paket data [RAD-99].

Filter dapat diatur untuk tiap-tiap kelas atau untuk tiap-tiap disiplin antrian berdasarkan dari desain dari disiplin antrian.

2.5.4 *Policing*

Fungsi dari *policing* adalah untuk memastikan bahwa trafik tidak melanngar batasan-batasan yang telah ditentukan.

BAB III

METODOLOGI PENELITIAN

Tahap ini akan menjelaskan langkah-langkah yang akan dilakukan untuk merancang sistem yang akan dibuat hingga dapat berfungsi sebagaimana yang diharapkan. Langkah-langkah tersebut adalah sebagai berikut :

31.. Studi Literatur

Studi literatur yang dilakukan bertujuan untuk mengkaji hal-hal yang berhubungan dengan teori-teori yang mendukung dalam perencanaan dan perancangan sistem, yaitu :

1. Kajian pustaka mengenai Konsep Jaringan Komputer, yaitu suatu cara untuk menghubungkan antara satu komputer dengan komputer lainnya menggunakan protokol komunikasi melalui media komunikasi sehingga dapat saling berbagi sumber daya yang ada.

2. Kajian pustaka mengenai protokol TCP/IP, yaitu suatu protokol yang digunakan oleh suatu komputer dalam suatu jaringan komputer agar dapat saling berkomunikasi dengan komputer lain.

3. Kajian pustaka mengenai *Quality of Service*, yaitu suatu pengukuran tentang seberapa baik jaringan dan merupakan suatu usaha untuk mendefinisikan karakteristik dan sifat dari suatu servis.

4. Kajian pustaka mengenai *Linux Traffic Control*, yaitu suatu metoda pengendalian *bandwidth* di Linux.

5. Kajian pustaka mengenai Disiplin Antrian, yaitu suatu metode untuk menentukan bagaimana data akan dikirimkan.

3.2 Perancangan dan Implementasi Sistem

Pada tahap ini, sistem untuk melakukan manajemen *bandwidth* dirancang sebagaimana berikut :

1. Perancangan pengujian *bandwidth manager*.
2. Perancangan jaringan komputer.
3. Pemasangan sistem operasi pada setiap komputer di jaringan komputer.
4. Pengkonfigurasian *bandwidth manager* menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB) secara bergantian sehingga dapat bekerja sesuai dengan yang diharapkan.

3.3 Pengujian dan Analisis Sistem

Pada tahap pengujian dan analisis sistem ini, *bandwidth manager* yang telah dibangun diuji dengan beberapa cara untuk mengetahui parameter waktu proses, *throughput*, *jitter*, dan *loss datagram* dengan pembedaan pada pembagian kapasitas *bandwidth*, nomor *port* dan alamat IP *address*. Hasil dari pengujian tersebut akan dianalisis untuk mengetahui jenis *queueing discipline* yang lebih baik dan optimal diantara keduanya yang bisa secara langsung diimplementasikan pada komputer *gateway* berbasis sistem operasi Linux.

3.4 Pengambilan Kesimpulan dan Saran

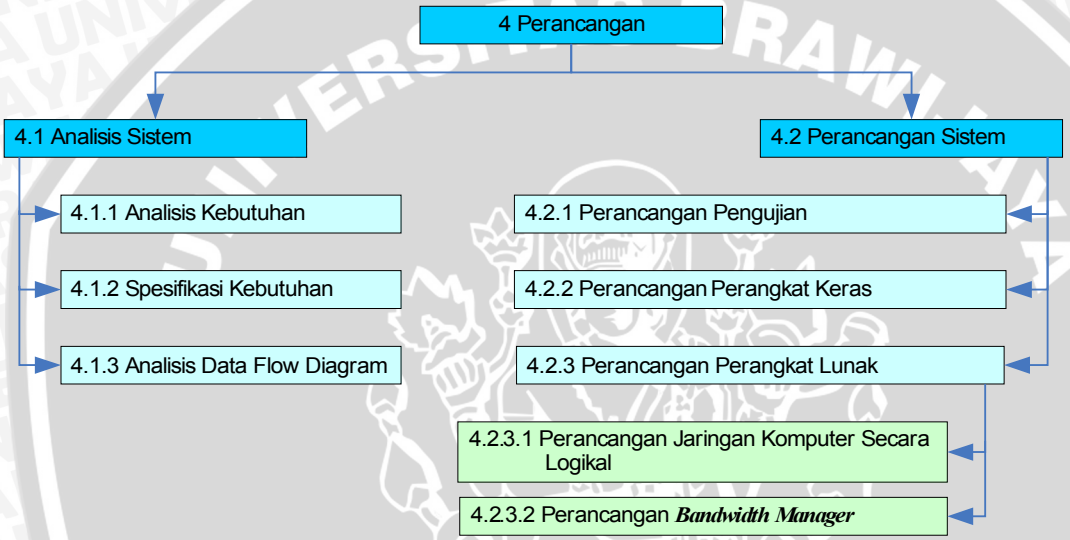
Tahap ini adalah tahap pengambilan kesimpulan dari sistem yang telah dibuat. Pengambilan kesimpulan dilakukan setelah semua tahap perancangan dan pengujian sistem telah selesai dilakukan dan didasarkan pada kekesuaina antara terori dan praktek. Kesimpulan ini merupakan informasi akhir dari pengujian sistem yang berisi mengenai kinerja sistem tersebut.

Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi serta menyempurnakan penulisan.

BAB IV

PERANCANGAN

Bab ini menjelaskan tentang perancangan *bandwidth manager* pada jaringan komputer menggunakan disiplin antrian *Class Based Queuing* (CBQ) atau menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB). Perancangan yang dilakukan dapat digambarkan dengan diagram pohon seperti dalam Gambar 4.1 berikut :



Gambar 4.1 Diagram Pohon Perancangan

Sumber: [Perancangan]

Perancangan *bandwidth manager* pada jaringan komputer menggunakan disiplin antrian *Class Based Queuing* (CBQ) atau menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB) dibagi menjadi dua tahap utama, yaitu : analisis sistem dan perancangan sistem. Analisis sistem yang dilakukan meliputi tiga tahap, yaitu analisis kebutuhan, spesifikasi kebutuhan, dan analisis *Data Flow Diagram* (DFD). Perancangan sistem yang dilakukan meliputi tiga tahap, yaitu perancangan pengujian, perancangan diagram jaringan komputer, dan perancangan perangkat lunak.

4.1 Analisis Sistem

Analisis sistem dibutuhkan untuk menggambarkan kebutuhan yang dibutuhkan untuk membangun dasar bagi pembuatan perancangan sistem. Proses analisis sistem

meliputi analisis kebutuhan, spesifikasi kebutuhan, dan analisis *Data Flow Diagram* (DFD).

4.1.1 Analisis Kebutuhan

Analisis kebutuhan dibutuhkan untuk menjadi dasar yang akurat bagi perancangan sistem yang akan dibuat serta menyediakan referensi bagi dilakukannya validasi sistem. Berdasarkan latar belakang dari penulisan tugas akhir ini didapatkan kebutuhan-kebutuhan yaitu :

1. Melakukan analisis *bandwidth manager* menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB) pada sistem operasi Linux.
2. Dari analisis ini diharapkan didapat jenis disiplin antrian yang paling baik dan optimal diantara keduannya yang bisa secara langsung diimplementasikan pada komputer *gateway* berbasis sistem operasi Linux.

4.1.2 Spesifikasi Kebutuhan

Spesifikasi kebutuhan dibutuhkan untuk menjelaskan kebutuhan yang telah didefinisikan pada subbab 4.1.1 secara lebih detail dan tepat yang akan menjadi dasar bagi perancangan dan implementasi.

Definisi :

1. Analisis *bandwidth manager* menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB) pada sistem operasi Linux.

Spesifikasi :

1.1 Merancang dan mengkonfigurasi Linux agar dapat berfungsi sebagai *bandwidth manager* menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB).

1.2 Merancang dan mengkonfigurasi Linux agar dapat berfungsi sebagai *bandwidth manager* berdasarkan pada kapasitas *bandwidth*, nomor *port* dan alamat IP *address*.

1.3 Menganalisis dan memonitoring jalannya paket data yang melewati komputer *bandwidth manager*.

Definisi :

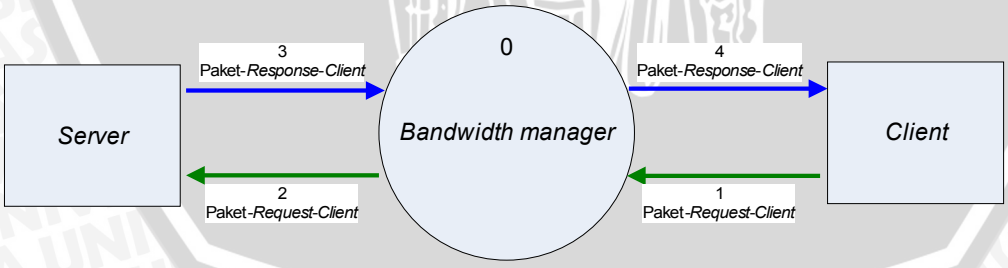
2. Dari analisis ini diharapkan didapat jenis disiplin antrian yang paling baik dan optimal diantara keduannya yang bisa secara langsung diimplementasikan pada komputer *gateway* berbasis sistem operasi Linux.

Spesifikasi :

2.1 Mengukur kinerja dan kemampuan *bandwidth manager* berdasarkan parameter waktu proses, *throughput*, *jitter*, dan *loss datagram* dengan pembedaan pada pembagian kapasitas *bandwidth*, nomor *port* dan alamat IP *address*.

4.1.3 Analisis *Data Flow Diagram*

DFD yang pertama kali dibuat adalah DFD level 0 atau *Context Diagram* atau Diagram Konteks. Diagram konteks merupakan diagram yang menampilkan masukan proses, proses dan keluaran proses dari sistem secara umum.



Gambar 4.2 Diagram Konteks *Bandwidth manager*

Sumber: [Perancangan]

Berdasarkan Gambar 4.2, proses sistem *bandwidth manager* memiliki beberapa tipe aliran data pada sistem yaitu berupa *request* data (garis yang berwarna hijau), dan *response* data (garis yang berwarna biru). Perbedaan warna garis aliran data pada

diagram konteks (Gambar 4.2) digunakan untuk memudahkan dalam menganalisis aliran data yang ada.

Diagram konteks sistem *bandwidth manager* ditunjukkan dalam Gambar 4.2. Berdasarkan Gambar 4.2 proses sistem *bandwidth manager* mempunyai masukan :

- Paket-*Request-Client*

  Paket-*Request-Client* merupakan paket data yang digunakan untuk meminta koneksi dari komputer *client* ke komputer *server*.

Berdasarkan Gambar 4.2 proses sistem *bandwidth manager* mempunyai keluaran :

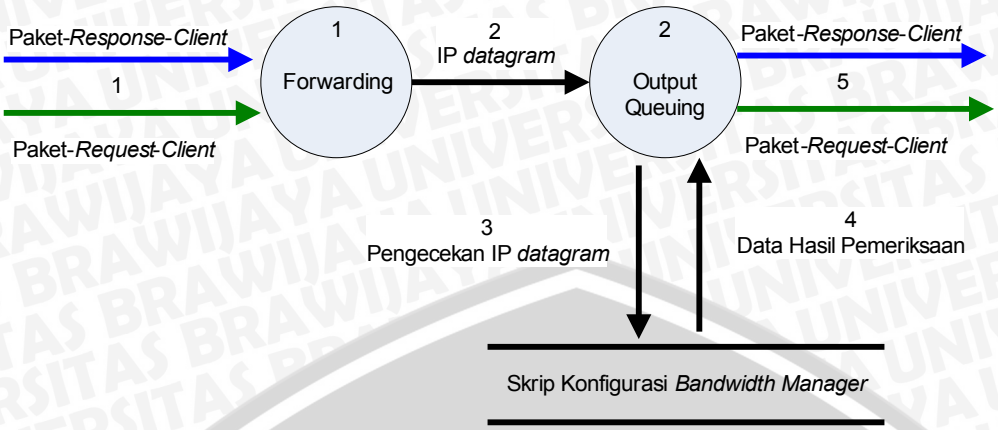
- Paket-*Response-Server*

  Paket-*Response-Server* merupakan data paket berisi *response* koneksi data dari permintaan koneksi *client*.

Komputer *client* untuk melakukan koneksi akan mengirim paket *Paket-Request-Client* ke komputer *server*. Paket ini akan melewati komputer *bandwidth manager* dan akan diteruskan oleh *bandwidth manager* ke komputer *server*. Kemudian komputer *server* akan mengirimkan paket Paket-*Response-Server* sebagai balasan terhadap permintaan dari komputer *client*. Paket ini akan melewati komputer *bandwidth manager* dan akan diteruskan oleh *bandwidth manager* ke komputer *client*.

4.1.3.1 *Data Flow Diagram* Level 0

Data Flow Diagram (DFD) level 1 merupakan penjabaran dari diagram konteks dimana pada level ini masih bisa dijabarkan lagi pada level berikutnya. DFD level 1 sistem ***b**andwidth manager* ditunjukkan dalam Gambar 4.3.



Gambar 4.3 DFD level 0 *Bandwidth manager*

Sumber: [Perancangan]

DFD level 0 memiliki 2 proses yaitu: Proses *Forwarding* dan Proses *Output Queueing*. Penjabaran proses-proses tersebut adalah :

1. Proses *Forwarding*

   Proses ini merupakan proses pemeriksaan paket IP *datagram* untuk menentukan jalur pengiriman paket IP *datagram*.

2. Proses *Output Queueing*

   Proses ini akan mengatur bagaimana paket IP datagram akan diteruskan sesuai dengan *rule bandwidth manager*.

4.2 Perancangan Sistem

Perancangan *bandwidth manager* pada jaringan komputer menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan *Hierarchical Token Bucket* (HTB) dibagi menjadi tiga tahap utama, yaitu perancangan pengujian, perancangan perangkat keras dan perancangan perangkat lunak. Perancangan pengujian adalah merancang skenario bagaimana *bandwidth manager* akan diimplementasikan dan diuji. Perancangan perangkat keras adalah merancang topologi jaringan komputer. Perancangan perangkat lunak meliputi : perancangan jaringan komputer secara *logical*, perancangan *bandwidth manager*.

4.2.1 Perancangan Pengujian Sistem

Pengujian pada *bandwidth manager* dilakukan sesuai dengan tujuan penulisan tugas akhir ini yaitu : mengimplementasikan serta membandingkan *bandwidth manager* menggunakan disiplin antrian Class *Based Queuing (CBQ)* dan *Hierarchical Token Bucket (HTB)* berdasarkan parameter waktu proses, *throughput*, *jitter*, dan *loss datagram* dengan pembedaan pada pembagian kapasitas *bandwidth*, nomor *port* dan alamat IP *address*.

Implementasi untuk pengujian ini dibagi ke dalam 3 skenario besar yaitu:

1. Skenario dengan *leaf class* berdasarkan IP *address*

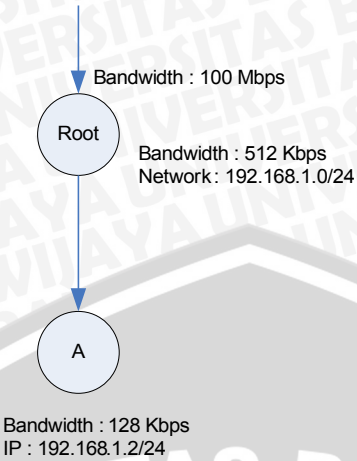
   Dalam skenario ini diperlihatkan cara mengalokasikan *bandwidth* berdasarkan pada IP *address*, dipilihnya IP *address* sebagai bahan perbandingan manajemen *bandwidth* CBQ dan HTB karena secara *de facto*, IP *address* dikenal dan digunakan secara luas di seluruh dunia sebagai protokol pada Internet.

   Untuk skenario manajemen *bandwidth* berdasarkan IP *address* dibagi dalam 4 skenario yaitu skenario dengan satu *leaf class* berdasarkan IP *address,* dua *leaf class* berdasarkan IP *address*, tiga *leaf class* berdasarkan IP *address* dan empat *leaf class* berdasarkan IP *address*.

   Pada *bandwidth manager* CBQ dan HTB, dialokasikan *bandwidth* sebesar : 512 Kbps untuk *network* 192.168.1.0/24 dan 128 Kbps untuk masing-masing *client*, kemudian dikirimkan data berukuran 1 MBytes secara bersamaan ke masing-masing IP *address*, masing-masing kelas yang berdasarkan IP tersebut bisa saling meminjam *bandwidth* saat *bandwidth* salah satu kelas sedang tidak menerima data/*idle*.

   a. Skenario dengan satu ***leaf class*** berdasarkan IP ***address.***

      Pada skenario ini, terdapat satu buah komputer *client* yang aktif terkoneksi ke komputer *server*.

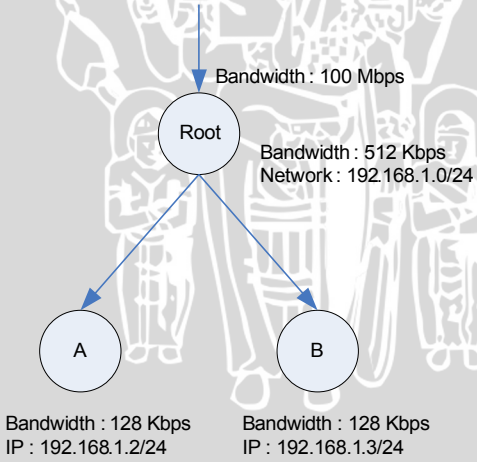


Gambar 4.4 *Link Sharing* dengan satu *leaf class* berdasarkan IP *address*

Sumber: [Perancangan]

b. Skenario dengan dua ***leaf class*** berdasarkan IP ***address.***

Pada skenario ini, terdapat dua buah komputer *client* yang aktif terkoneksi ke komputer *server*.

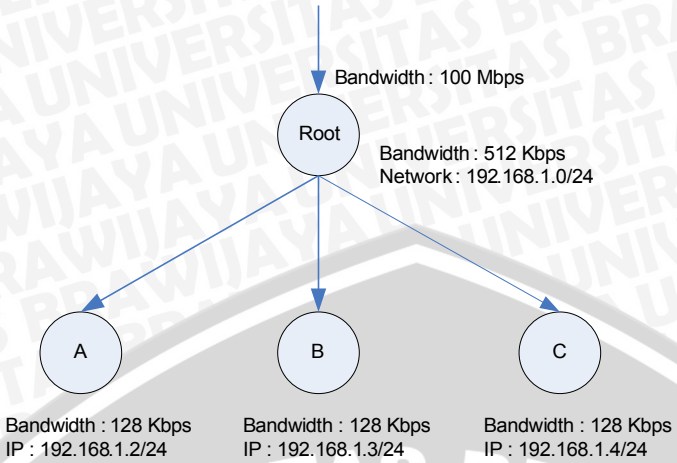


Gambar 4.5 *Link Sharing* dengan dua *leaf class* berdasarkan IP *address*

Sumber: [Perancangan]

c. Skenario dengan tiga *leaf class* berdasarkan IP *address*

Pada skenario ini, terdapat tiga buah komputer *client* yang aktif terkoneksi ke komputer *server*.

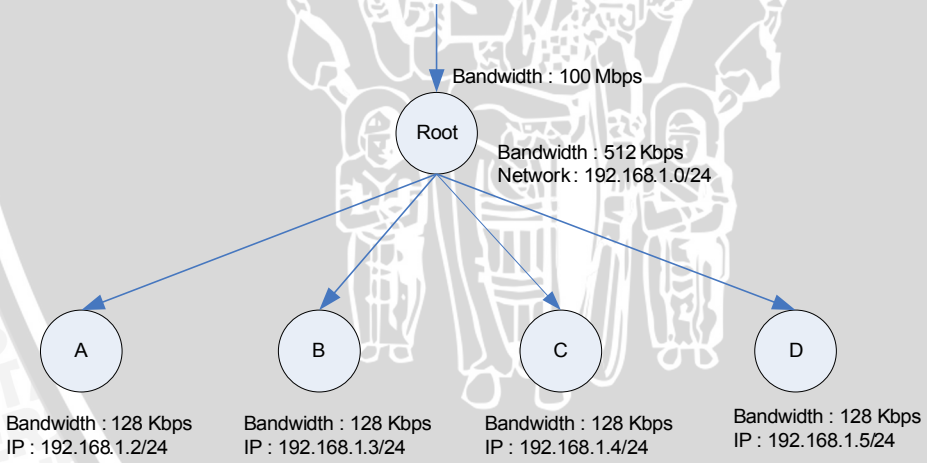


Gambar 4.6 *Link Sharing* dengan tiga *leaf class* berdasarkan IP *address*

Sumber: [Perancangan]

d. Skenario dengan empat *leaf class* berdasarkan IP *address*

Pada skenario ini, terdapat empat buah komputer *client* yang aktif terkoneksi ke komputer *server*.



Gambar 4.7 *Link Sharing* dengan empat *leaf class* berdasarkan IP *address*

Sumber: [Perancangan]

2. Skenario dengan *leaf class* berdasarkan *port*

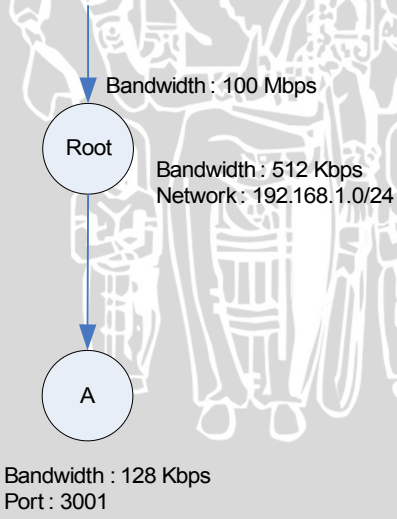
Dalam skenario ini diperlihatkan cara mengalokasikan *bandwidth* berdasarkan pada nomor *port*. *Port* pada protokol TCP/IP mencerminkan jenis layanan/aplikasi yang digunakan.

Untuk skenario manajemen *bandwidth* berdasarkan *port* dibagi dalam 4 skenario yaitu skenario dengan satu *leaf class* berdasarkan *port*, dua *leaf class* berdasarkan *port*, tiga *leaf class* berdasarkan *port* dan empat *leaf class* berdasarkan *port*.

Pada *bandwidth manager* CBQ dan HTB, dialokasikan *bandwidth* sebesar : 512 Kbps untuk *network* 192.168.1.0/24 dan 128 Kbps untuk masing-masing *port*, kemudian dikirimkan data berukuran 1 MBytes secara bersamaan ke masing-masing nomor *port*, masing-masing kelas yang berdasarkan *port* tersebut bisa saling meminjam *bandwidth* saat *bandwidth* salah satu kelas sedang tidak menerima data/*idle*.

a. Skenario dengan satu *leaf class* berdasarkan *port*

Pada skenario ini, terdapat satu buah komputer *client* yang memiliki satu koneksi aktif ke komputer *server*.

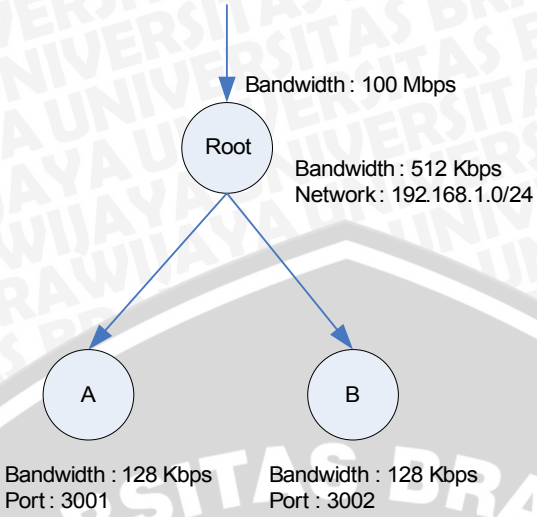


Gambar 4.8 *Link Sharing* dengan satu *leaf class* berdasarkan *port*

Sumber: [Perancangan]

b. Skenario dengan dua *leaf class* berdasarkan *port*

Pada skenario ini, terdapat satu buah komputer *client* yang memiliki dua koneksi aktif ke komputer *server*.

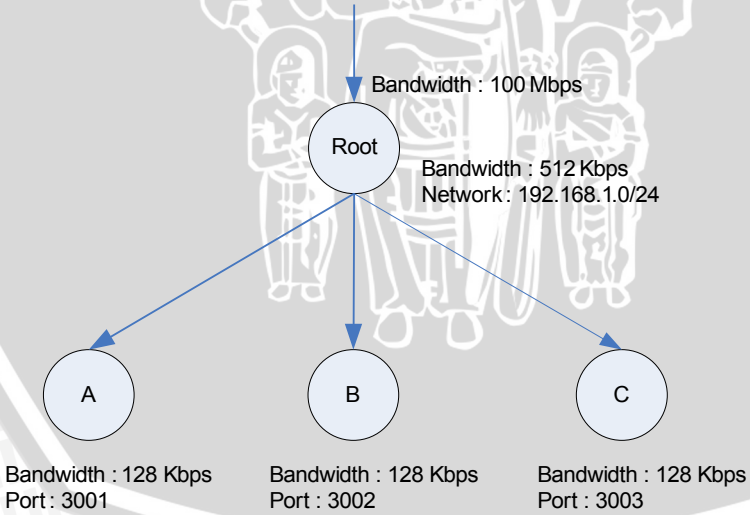


Gambar 4.9 *Link Sharing* dengan dua *leaf class* berdasarkan *port*

Sumber: [Perancangan]

c. Skenario dengan tiga *leaf class* berdasarkan *port*

Pada skenario ini, terdapat satu buah komputer *client* yang memiliki tiga koneksi aktif ke komputer *server*.

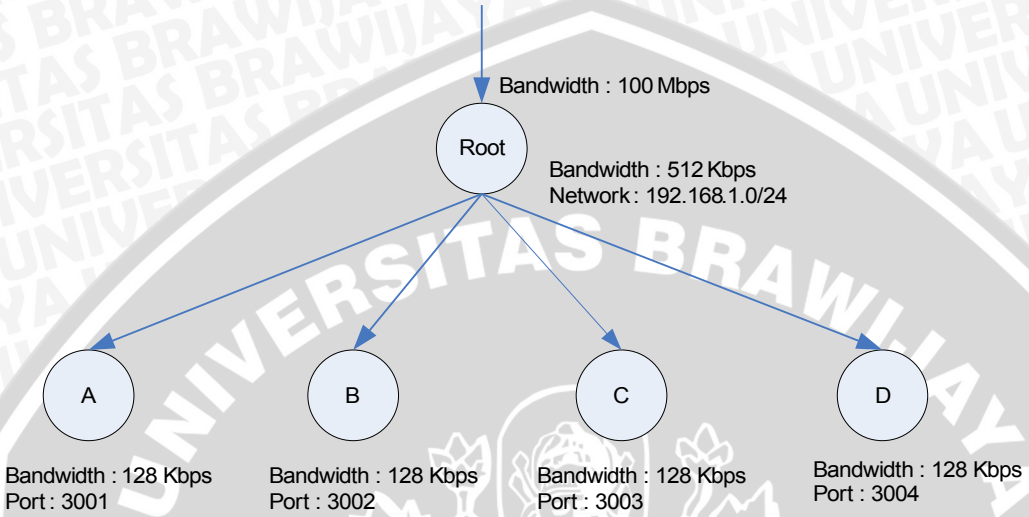


Gambar 4.10 *Link Sharing* dengan tiga *leaf class* berdasarkan *port*

Sumber: [Perancangan]

d. Skenario dengan empat *leaf class* berdasarkan *port*

Pada skenario ini, terdapat satu buah komputer *client* yang memiliki empat koneksi aktif ke komputer *server*.

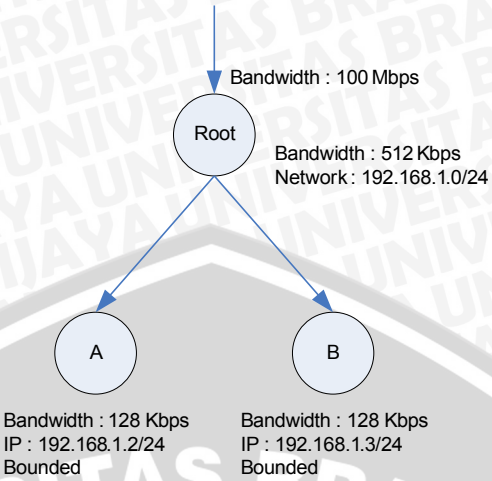


Gambar 4.11 *Link Sharing* dengan empat *leaf class* berdasarkan *port*

Sumber: [Perancangan]

3. Skenario dengan parameter "tidak meminjamkan *bandwidth idle*"

Skenario ini menunjukkan penggunaan parameter yang mematikan fungsi *link sharing* berarti kelas-kelas yang sedang melayani antrian hanya dapat menggunakan *bandwidth* sebatas pada alokasinya masing-masing, *bandwidth* kelas lain yang *idle* tidak dapat digunakan untuk meningkatkan *rate bandwidth* kelas yang sedang menerima antrian pesan. Skenario ini pada dasarnya membagi *link* menjadi dua kelas berdasarkan IP Address dengan alokasi *bandwidth*-nya sebagai berikut IP 192.168.1.2 dengan *bandwidth* 128 kbps dan IP 192.168.1.3 mendapat alokasi *rate* 128 kbps.

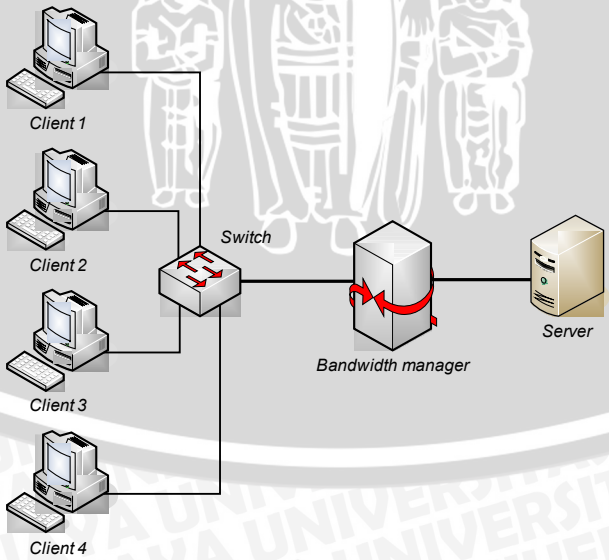


Gambar 4.12 *Link Sharing* dengan dua *leaf class* dengan parameter "tidak meminjamkan *bandwidth idle*"

Sumber: [Perancangan]

4.2.2 Perancangan Perangkat Keras

Perancangan perangkat keras adalah merancang suatu topologi jaringan komputer tempat sistem yang akan diimplementasikan. Topologi jaringan komputer yang digunakan pada sistem ini adalah topologi *star*, sebagaimana yang diperlihatkan dalam Gambar 4.13.



Gambar 4.13 Diagram Jaringan dengan *bandwidth manager*

Sumber: [Perancangan]

Dalam Gambar 4.13 di atas, dapat dijelaskan bahwa komputer *client* sebanyak 4 buah terhubung ke sebuah *switch*. Selanjutnya *switch* tersebut terhubung ke sebuah komputer *server* melalui komputer *bandwidth manager*.

Perangkat keras yang dibutuhkan untuk implementasi jaringan komputer ini antara lain :

- 4 unit komputer, masing-masing dengan spesifikasi minimal Pentium 1 200 Mhz, memori minimal 64 MB, 1 buah NIC 100 Mbps, dan Hardisk minimal 4 GB yang akan digunakan sebagai *client*.
- 1 unit komputer dengan spesifikasi minimal Pentium III 500 Mhz , memori minimal 128 MB, 2 buah NIC 100 Mbps dan Hardisk minimal 10 GB yang akan digunakan sebagai *bandwidth manager*. *Bandwidth manager* ini bisa difungsikan sebagai *router*.
- 1 unit komputer dengan spesifikasi minimal Pentium III 500 Mhz , memori minimal 128 MB, 1 buah NIC 100 Mbps, dan Hardisk minimal 10 GB yang akan digunakan sebagai *server*.
- 1 buah switch 8 *port* 100 Mbps.
- Kabel UTP category 5 sebagai media transmisi untuk hubungan antar komputer.

4.2.3 Perancangan Perangkat Lunak

Perangkat lunak yang akan diimplementasikan dirancang dengan beberapa tahap secara berurutan, yaitu : perancangan jaringan komputer secara *logical* dan perancangan *bandwidth manager*.

4.2.3.1 Perancangan Jaringan Komputer Secara *Logical*

Topologi jaringan komputer yang secara fisik telah dirancang tersebut, perlu pula dirancang secara *logical*. Perancangan secara *logical* adalah perancangan jaringan komputer dengan cara melakukan konfigurasi pada perangkat lunak dari perangkat-perangkat yang terhubung di jaringan komputer sehingga perangkat-perangkat tersebut

dapat saling berkomunikasi. Konfigurasi tersebut adalah melakukan pengalamatan dengan IP *address*.

Komputer *server* akan diberi konfigurasi sebagai berikut :

- IP *address* : 192.168.0.2
- *Subnet Mask* : 255.255.255.0
- *Default gateway* : 192.168.0.1

Sedangkan komputer *client* masing-masing akan diberi konfigurasi sebagai berikut :

- *Client* 1
    - IP *address* : 192.168.1.2
    - *Subnet Mask* : 255.255.255.0
    - *Default gateway* : 192.168.1.1
- *Client* 2
    - IP *address* : 192.168.1.3
    - *Subnet Mask* : 255.255.255.0
    - *Default gateway* : 192.168.1.1
- *Client* 3
    - IP *address* : 192.168.1.4
    - *Subnet Mask* : 255.255.255.0
    - *Default gateway* : 192.168.1.1

- *Client* 4
    - IP *address* : 192.168.1.5
    - *Subnet Mask* : 255.255.255.0
    - *Default gateway* : 192.168.1.1

Sistem operasi yang digunakan untuk *server* dan *client* adalah sistem operasi Linux distribusi Slackware versi 11. Perangkat Lunak untuk *client* dan *server* adalah `Iperf`. `Iperf` merupakan suatu perangkat lunak yang digunakan untuk mengukur performansi jaringan.

4.2.3.2 Perancangan *Bandwidth Manager*

Komputer *bandwidth manager* adalah komputer yang berfungsi sebagai pengatur kecepatan bagi suatu paket data saat memasuki network tertentu. *Bandwidth manager* dapat berupa sebuah *router*.

Sistem operasi yang digunakan untuk *bandwidth manager* adalah sistem operasi Linux distribusi Slackware versi 11. Perangkat Lunak untuk *bandwidth manager*:

- `GCC`, perangkat lunak yg digunakan untuk melakukan kompilasi kernel.
- Editor `Vim`, yaitu perangkat lunak yg digunakan untuk membuat dan mengedit file konfigurasi.
- `Iproute2`, perangkat lunak yg digunakan untuk melakukan *routing* dan manajemen *bandwidth*.

Setelah Sistem Operasi Slackware 11 di-*install*, maka agar komputer *bandwidth manager* berfungsi sebagai *bandwidth manager*, perlu dikonfigurasi sebagai berikut :

- Mengaktifkan fungsi untuk meneruskan paket data.
- Pada eth0, sebagai *uplink* menuju ke komputer *server*, dikonfigurasi sebagai berikut :

- IP *address* : 192.168.0.1
- *Subnet Mask* : 255.255.255.0

- Pada eth1, sebagai *downlink* menuju ke komputer *client*, dikonfigurasi sebagai berikut :
  - IP *Address* : 192.168.1.1
  - *Subnet Mask* : 255.255.255.0

- Mengkonfigurasi kernel Linux untuk melakukan manajemen *bandwidth* menggunakan `Iproute2` dengan bantuan file skrip.

BAB V

IMPLEMENTASI

Bandwidth manager pada jaringan komputer menggunakan disiplin antrian *Class Based Queuing* (CBQ) atau menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB) yang telah dirancang pada Bab Empat, akan diimplementasikan pada tahap ini. Ada dua tahap utama yang dilakukan pada bagian implementasi sistem ini, yaitu implementasi jaringan komputer dan implementasi *bandwidth manager* menggunakan disiplin antrian *Class Based Queuing* (CBQ) atau menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB).

5.1 Implementasi Jaringan Komputer

Pada tahap ini, rancangan jaringan komputer yang telah dibuat pada bab sebelumnya akan diwujudkan. Jaringan komputer terdiri dari dua bagian besar, yaitu bagian perangkat keras dan bagian perangkat lunak.

5.1.1 Perangkat Keras Jaringan Komputer

Pada bagian perangkat keras, komputer yang akan berfungsi sebagai *bandwidth manager* memiliki spesifikasi sebagai berikut :

- *Processor* : Intel Pentium III – 1 GHz
- *Motherboard* : Shuttle MV25
- *Harddisk* : Quantum Fireball 10 GB – 5400 RPM
- *Memory* : Visipro 256 MB PC 100
- LAN *Card* : 2 x 3Com 3C905B
- *Graphic Card* : S3 Trio 3D/2X 8 MB

Sedangkan komputer yang berfungsi sebagai *client* memiliki spesifikasi :

- *Processor* : Intel Pentium III – 350 MHz

- *Motherboard* : Asus P3V4X
- *Harddisk* : Quantum Fireball 10 GB – 5400 RPM
- *Memory* : Visipro 128 MB PC 100
- LAN *Card* : AMD PCNet 32
- *Graphic Card* : SiS 6326 4 MB

Dan komputer yang berfungsi sebagai *server* memiliki spesifikasi :

- *Processor* : Intel Pentium III – 550 MHz
- *Motherboard* : Asus P3V4X
- *Harddisk* : Quantum Fireball 10 GB – 5400 RPM
- *Memory* : Visipro 256 MB PC 100
- LAN *Card* : 3Com 3C905B
- *Graphic Card* : SiS 6326 4 MB

Perangkat keras lainnya adalah sebagai berikut :

- *Switch* : *Switch Manageable* 3Com 3C17300A 24 *port*
- Kabel UTP : kabel Belden CDT Networking
- Konektor : RJ-45 AMP buatan Tyco Corporation

Perangkat keras tersebut kemudian disusun sehingga membentuk sebuah jaringan komputer dengan Topologi *Star* seperti yang diperlihatkan dalam Gambar 4.13.

5.1.2 Perangkat Lunak Jaringan Komputer

Pada tahap ini perangkat keras yang telah disusun, dikonfigurasi perangkat lunaknya. Perangkat keras yang dikonfigurasi adalah komputer *bandwidth manager*, *client*, dan *server*.

5.1.2.1 Konfigurasi Komputer *Bandwidth Manager*

Konfigurasi pada komputer *bandwidth manager* dilakukan setelah Sistem Operasi Slackware 11 ter-*install*. Semua konfigurasi dilakukan pada *Terminal – Command Line*.

Untuk melakukan konfigurasi, maka perlu *login* sebagai root.

Konfigurasi yang perlu dilakukan agar komputer *bandwidth manager* dapat berfungsi dengan baik adalah sebagai berikut :

a. Konfigurasi eth0 sebagai *uplink* dan eth1 sebagai *downlink*

Konfigurasi ini dilakukan dengan melakukan perubahan pada *file* `rc.inet1.conf` seperti dibawah ini :

```
# vim /etc/rc/d/rc.inet1.conf
```

```
# Config information for eth0:
IPADDR[0]="192.168.0.1"
NETMASK[0]="255.255.255.0"
USE_DHCP[0]="no"
DHCP_HOSTNAME[0]=""

# Config information for eth1:
IPADDR[1]="192.168.1.1"
NETMASK[1]="255.255.255.0"
USE_DHCP[1]="no"
DHCP_HOSTNAME[1]=""
```

b. Mengaktifkan fungsi meneruskan paket data

Konfigurasi ini dilakukan dengan mengubah hak akses dan mode dari *file* `rc.ip_forward` menjadi dapat dieksekusi seperti dibawah ini :

```
# chmod 755 etc/rc/d/rc.ip_forward
```

```
#!/bin/sh
# /etc/rc.d/rc.ip_forward:  start/stop IP packet forwarding
#
# If you intend to run your Linux box as a router, i.e. as a
# computer that forwards and redistributes network packets, you
# will need to enable IP packet forwarding in your kernel.
#
# To activate IP packet forwarding at boot time, make this
# script executable:  chmod 755 /etc/rc.d/rc.ip_forward
#
# To disable IP packet forwarding at boot time, make this
# script non-executable:  chmod 644 /etc/rc.d/rc.ip_forward

# Start IP packet forwarding:
ip_forward_start() {
  if [ -f /proc/sys/net/ipv4/ip_forward ]; then
    echo "Activating IPv4 packet forwarding."
    echo 1 > /proc/sys/net/ipv4/ip_forward
  fi
  # When using IPv4 packet forwarding, you will also get the
  # rp_filter, which automatically rejects incoming packets if the
  # routing table entry for their source address doesn't match the
  # network interface they're arriving on.  This has security
  # advantages because it prevents the so-called IP spoofing,
  # however it can pose problems if you use asymmetric routing
  # (packets from you to a host take a different path than packets
  # from that host to you) or if you operate a non-routing host
  # which has several IP addresses on different interfaces.  To
  # turn rp_filter off, uncomment the lines below:
  #if [ -r /proc/sys/net/ipv4/conf/all/rp_filter ]; then
  #  echo "Disabling rp_filter."
  #  echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter
  #fi
}

# Stop IP packet forwarding:
ip_forward_stop() {
  if [ -f /proc/sys/net/ipv4/ip_forward ]; then
    echo "Disabling IPv4 packet forwarding."
    echo 0 > /proc/sys/net/ipv4/ip_forward
  fi
}

# Restart IP packet forwarding:
ip_forward_restart() {
  ip_forward_stop
  sleep 1
  ip_forward_start
}

case "$1" in
'start')
  ip_forward_start
  ;;
'stop')
  ip_forward_stop
  ;;
'restart')
  ip_forward_restart
  ;;
*)
  echo "usage $0 start|stop|restart"
esac
```

5.1.2.2 Konfigurasi Komputer *Client*

Konfigurasi pada komputer *client* dilakukan setelah Sistem Operasi Slackware 11 ter-*install*. Semua konfigurasi dilakukan pada *Terminal – Command Line*.

Untuk melakukan konfigurasi, maka perlu *login* sebagai root.

Konfigurasi yang perlu dilakukan agar komputer *client* dapat berfungsi dengan baik adalah mengonfigurasi eth0 sebagai *uplink* dan memasang *Default Gateway*. Konfigurasi ini dilakukan dengan melakukan perubahan pada *file* `rc.inet1.conf`. Contoh konfigurasi pada salah satu komputer *client* adalah seperti di bawah ini :

```
# vim /etc/rc/d/rc.inet1.conf
```

```
# Config information for eth0:
IPADDR[0]="192.168.1.2"
NETMASK[0]="255.255.255.0"
USE_DHCP[0]="no"
DHCP_HOSTNAME[0]=""

# Default gateway IP address:
GATEWAY="192.168.1.1"
```

5.1.2.3 Konfigurasi Komputer *Server*

Konfigurasi pada komputer *server* dilakukan setelah Sistem Operasi Slackware 11 ter-*install*. Semua konfigurasi dilakukan pada *Terminal – Command Line*.

Untuk melakukan konfigurasi, maka perlu *login* sebagai root.

Konfigurasi yang perlu dilakukan agar komputer *server* dapat berfungsi dengan baik adalah mengonfigurasi eth0 sebagai *uplink* dan memasang *Default Gateway*. Konfigurasi ini dilakukan dengan melakukan perubahan pada *file* `rc.inet1.conf` seperti di bawah ini :

```
# vim /etc/rc/d/rc.inet1.conf
```

```
# Config information for eth0:
IPADDR[0]="192.168.0.2"
NETMASK[0]="255.255.255.0"
USE_DHCP[0]="no"
DHCP_HOSTNAME[0]=""

# Default gateway IP address:
GATEWAY="192.168.0.1"
```

5.2 Implementasi *Bandwidth Manager*

Pada tahap ini, rancangan sistem yang telah dibuat diimplementasikan pada komputer *bandwidth manager* yang telah dikonfigurasi pada tahap sebelumnya. Implementasi sistem yang dibuat meliputi dua bagian utama yaitu, konfigurasi kernel Linux dan implementasi disiplin antrian.

5.2.1 Konfigurasi Kernel Linux

Kernel merupakan inti dari suatu sistem operasi, kernel berhubungan langsung dengan perangkat keras, untuk implementasi *bandwidth manager* lebih baik menggunakan versi kernel dengan versi yang paling baru atau setidak-tidaknya versi kernel 2.4.20 ke atas.

Diperlukan kompilasi kernel dengan tujuan mendapatkan versi kernel terbaru yang mendukung penerapan implementasi *bandwidth manager* CBQ dan HTB.

Langkah-langkah untuk mengkompilasi kernel adalah sebagai berikut :

1. Menyalin sumber kernel versi 2.6.17.13 dari media penyimpanan dan menempatkan pada direktori `/usr/src`

```
# cp linux-2.6.17.13.tar.bz2 /usr/src
```

2. Masuk ke direktori /usr/src

```
# cd /usr/src
```

3. Mengekstrak *file* kernel terkompresi tersebut

```
# tar jxvf linux-2.6.17.13.tar.bz2
```

4. Masuk ke direktori kernel linux-2.6.17.13

```
# cd /usr/src/linux-2.6.17.13
```

5. Memulai mengaktifkan option-option yang diperlukan

```
# make menuconfig
```

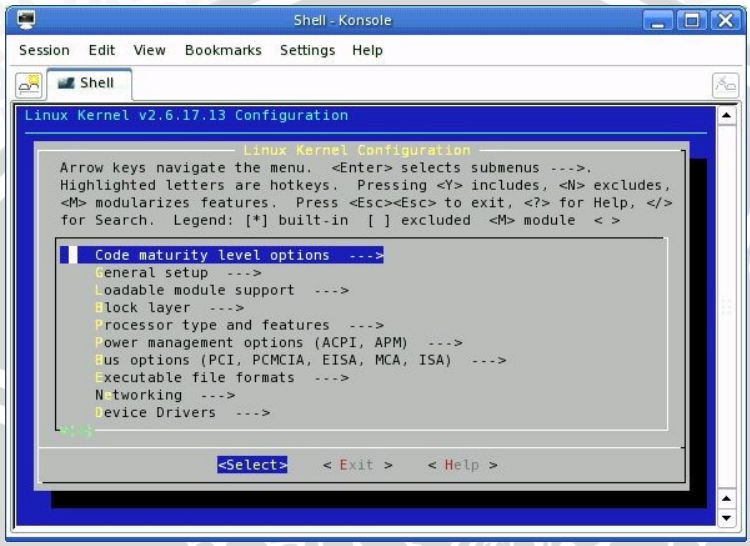
6. Setelah perintah tersebut dieksekusi, akan muncul menu-menu yang menunjukkan option konfigurasi kernel.

Option-option ini bisa diisi dengan :

‘Y’ : jika hendak mengaktifkan option tersebut .

‘N’ : jika tidak menginginkan mengaktifkan option tersebut .

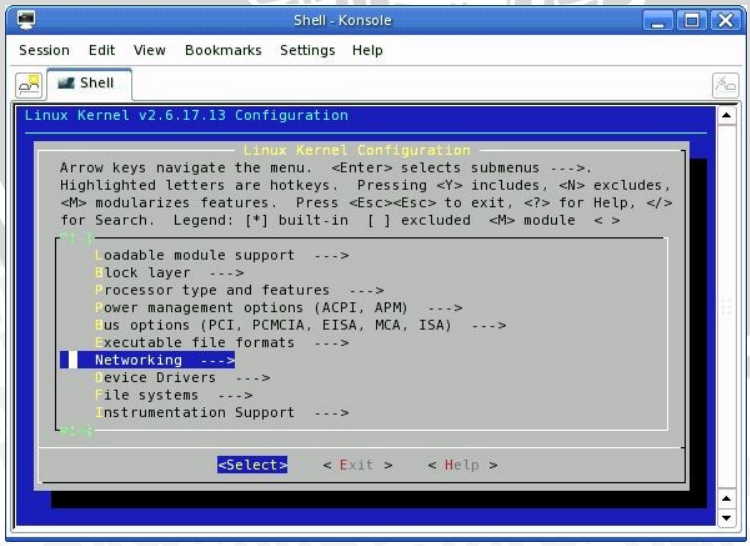
‘M’ : jika hendak mengkompilasi option dalam bentuk module.



Gambar 5.1 Konfigurasi kernel Linux

Sumber: [Implementasi]

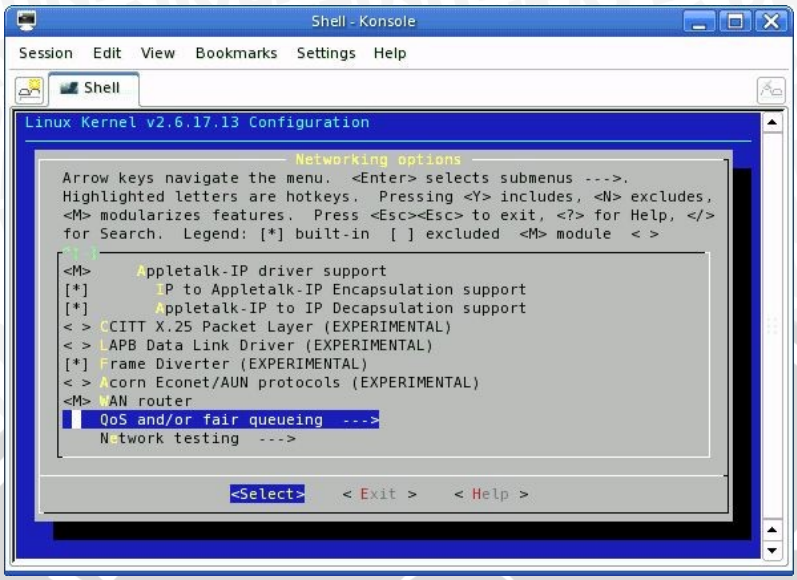
7. Dari menu-menu yang ditampilkan, pilih *option networking* :



Gambar 5.2 *Option Networking* di konfigurasi kernel Linux

Sumber: [Implementasi]

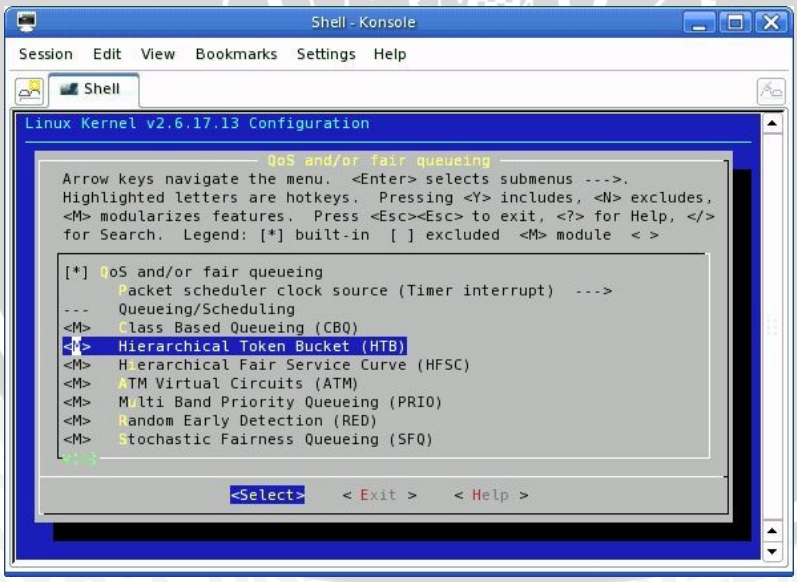
8. Dari menu-menu yang ditampilkan, pilih *option QoS and/or fair queuing* :



Gambar 5.3 *Option QoS and/or fair queuing* di konfigurasi kernel Linux

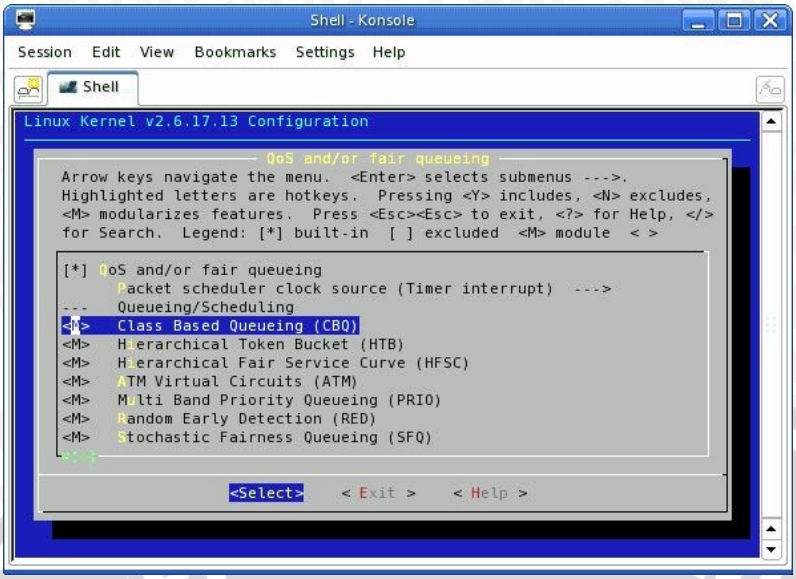
Sumber: [Implementasi]

8. Dari menu-menu yang ditampilkan, pilih *option Class Based Queuing* (CBQ) dan *option Hierarchical Token Bucket* (HTB) :



Gambar 5.4 *Option Hierarchical Token Bucket* (HTB) di konfigurasi kernel Linux

Sumber: [Implementasi]



Gambar 5.5 *Option Class Based Queuing* (CBQ) di konfigurasi kernel Linux

Sumber: [Implementasi]

9. Memeriksa *file* konfigurasi yang akan dikompilasi/diaktifkan

```
# make dep
```

10. Menghapus jejak hasil kompilasi sebelumnya

```
# make clean
```

11. Membuat hasil kompilasi sesungguhnya, bila tidak didapatkan *file* bzImage maka terjadi kesalahan kompilasi

```
# make bzimage
```

12. Modul-modul yang berhubungan dengan kompilasi kernel diinstal

```
# make module
# make modules_install
```

13. Menyalin hasil kompilasi kernel pada sistem

```
#cp /usr/src/linux-2.6.17.13/System.map /boot/
#cp /usr/src/linux-2.6.17.13/arch/i386/boot/bzImage /boot/vmlinuz-2.6.17.13
```

14. Mengubah isi tampilan *file* lilo.conf dengan menambahkan beberapa baris script:

```
# vim /etc/lilo.conf
```

```
# Linux bootable partition config begins
image= /boot/vmlinuz-2.6.17.13
    root= /dev/hda2
    label= Linux
    read-only
# Linux bootable partition config ends
```

15. Mengaktifkan *file* lilo yang telah ditambahkan beberapa script

```
# lilo
```

16. *Reboot* komputer

5.2.2 Implementasi Manajemen *Bandwidth*

Implementasi *bandwidth manager* dilakukan setelah kernel Linux dapat berfungsi dengan baik. Setelah tahap konfigurasi kernel Linux selesai, *bandwidth manager* dapat dikonfigurasi agar dapat bekerja. Pengkonfigurasian *bandwidth manager* dilakukan dengan perangkat lunak `Iproute2`. Untuk mempermudah implementasi *bandwidth manager* digunakan sebuah file skrip. Untuk disiplin antrian *Class Based Queuing* (CBQ) digunakan skrip `cbq.init` dan untuk disiplin antrian *Hierarchical Token Bucket* (HTB) digunakan skrip `htb.init`. Kedua skrip ini dapat dilihat lembar Lampiran A. Konfigurasi-konfigurasi yang dilakukan adalah seperti berikut:

a. Menyalin skrip `htb.init` dan `cbq.init` ke `/usr/rc.d/` dengan nama `rc.htb` dan `rc.cbq`

```
# cp htb.init /etc/rc.d/rc.htb
# cp cbq.init /etc/rc.d/rc.cbq
```

b. Membuat direktori tempat skrip *rule* limitasi *bandwidth* diletakkan. Skrip *rule* untuk CBQ diletakkan di direktori `/usr/sysconfig/cbq` dan skrip *rule* untuk HTB diletakkan di direktori `/usr/sysconfig/htb`.

```
# mkdir /etc/sysconfig/htb
# mkdir /etc/sysconfig/cbq
```

c. Membuat skrip *rule* dari skenario pengujian. Skrip untuk implementasi skenario *bandwidth manager* ini dapat dilihat di lembar Lampiran B.

d. Menjalankan salah satu antara disiplin antrian CBQ atau HTB untuk diuji. Disiplin antrian CBQ dijalankan dengan perintah :

```
# /etc/rc.d/rc.cbq start
```

Disiplin antrian HTB dijalankan dengan perintah :

```
# /etc/rc.d/rc.htb start
```

BAB VI

PENGUJIAN DAN ANALISIS

Sistem manajemen *bandwidth* yang telah diimplementasikan pada Bab Lima akan diuji dan dianalisis pada tahap ini. Pengujian dilakukan untuk mengetahui bagaimana kinerja dan kemampuan dari *bandwidth manager* menggunakan disiplin antrian *Class Based Queuing* (CBQ) dan menggunakan disiplin antrian *Hierarchical Token Bucket* (HTB) pada sistem operasi Linux. pengujian dan analisis dilakukan sesuai dengan skenario yang telah dibuat di Bab Empat.

Pengujian dilakukan dengan perangkat lunak `iperf`. `iperf` merupakan suatu perangkat lunak yang digunakan untuk mengukur performansi jaringan. Performasi yang diukur oleh `iperf` adalah waktu pengiriman, *throughput*, *jitter* dan *loss datagram*. Prosedur pengujian adalah :

1. Mengkonfigurasi *bandwidth manager* sesuai dengan skenario yang telah ditentukan.
2. Menjalankan skrip *bandwidth manager*.
3. Menjalankan `iperf` di komputer *client* sebagai *server* yang menerima paket data.
4. Menjalankan `iperf` di komputer *server* sebagai *client* yang mengirim paket data dengan mengirimkan data sebesar 1 MB.
5. Mencatat hasil pengujian yang tertera di layar komputer *client*.
6. Mengulangi pengujian ini sebanyak tiga kali.
7. Menghitung hasil rata-rata dari hasil pengujian.
8. Melakukan analisis dan mengambil kesimpulan dari hasil pengujian.

6.1 Pengujian

6.1.1 Skenario Dengan *Leaf Class* Berdasarkan IP *Address*

Tabel-Tabel selanjutnya merupakan representasi parameter rata-rata yang diukur. Untuk hasil pengujian secara lengkap, dapat dilihat di lembar Lampiran C.

IP *address*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
192.168.1.2	1	17,267	498,333	23,801	0/732

Tabel 6.1 Nilai rata-rata Parameter terukur HTB dengan 1 IP *address*

IP *address*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
192.168.1.2	1	16,667	516,333	12,551	0/732

Tabel 6.2 Nilai rata-rata Parameter terukur CBQ dengan 1 IP *address*

IP *address*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
192.168.1.2	1	33,667	255,667	45,456	0/732
192.168.1.3	1	34,033	253,333	27,162	0/732

Tabel 6.3 Nilai rata-rata Parameter terukur HTB dengan 2 IP *address*

IP *address*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
192.168.1.2	1	32,000	261,000	66,304	15/732
192.168.1.3	1	31,300	258,667	8,826	50/732

Tabel 6.4 Nilai rata-rata Parameter terukur CBQ dengan 2 IP *address*

IP *address*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
192.168.1.2	1	50,167	171,667	72,276	0/732
192.168.1.3	1	51,033	168,667	53,339	0/732
192.168.1.4	1	51,100	168,333	44,748	0/732

Tabel 6.5 Nilai rata-rata Parameter terukur HTB dengan 3 IP *address*

IP *address*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
192.168.1.2	1	32,367	188,000	68,826	213/732
192.168.1.3	1	31,500	175,000	38,923	263/732
192.168.1.4	1	29,133	176,000	7,865	296/732

Tabel 6.6 Nilai rata-rata Parameter terukur CBQ dengan 3 IP *address*

IP *address*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
192.168.1.2	1	63,133	136,333	75,827	0/732
192.168.1.3	1	64,067	134,667	58,530	0/732
192.168.1.4	1	64,033	134,667	52,224	0/732
192.168.1.5	1	49,867	125,000	97,183	42/732

Tabel 6.7 Nilai rata-rata Parameter terukur HTB dengan 4 IP *address*

IP *address*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
192.168.1.2	1	33,100	149,000	114,430	305/732
192.168.1.3	1	31,433	142,333	26,104	347/732
192.168.1.4	1	26,467	134,667	7,374	429/732
192.168.1.5	1	29,600	135,333	38,831	387/732

Tabel 6.8 Nilai rata-rata Parameter terukur CBQ dengan 4 IP *address*

6.1.2 Skenario Dengan *Leaf Class* Berdasarkan *Port*

Tabel-Tabel selanjutnya merupakan representasi parameter rata-rata yang diukur. Untuk hasil pengujian secara lengkap, dapat dilihat di lembar Lampiran C.

Nomor *port*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
3001	1	17,300	498,000	24,087	0/732

Tabel 6.9 Nilai rata-rata Parameter terukur HTB dengan 1 *port*

Nomor *port*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
3001	1	16,700	516,000	12,704	0/732

Nomor *port*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
3001	1	33,900	254,000	48,687	0/732
3002	1	34,233	251,667	29,808	0/732

Tabel 6.11 Nilai rata-rata Parameter terukur HTB dengan 2 *port*

Nomor *port*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
3001	1	31,100	257,333	11,180	52/732
3002	1	32,000	261,667	58,015	14/732

Tabel 6.12 Nilai rata-rata Parameter terukur CBQ dengan 2 *port*

Nomor *port*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
3001	1	50,200	171,667	69,046	0/732
3002	1	51,167	168,000	53,234	0/732
3003	1	51,033	168,667	41,809	0/732

Tabel 6.13 Nilai rata-rata Parameter terukur HTB dengan 3 *port*

Nomor *port*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
3001	1	30,933	188,333	36,154	231/732
3002	1	31,467	175,667	48,058	255/732
3003	1	29,700	183,667	14,442	267/732

Tabel 6.14 Nilai rata-rata Parameter terukur CBQ dengan 3 *port*

Nomor *port*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
3001	1	66,067	130,333	105,604	0/732
3002	1	67,767	127,000	78,316	0/732
3003	1	67,933	126,667	68,643	0/732
3004	1	67,733	127,000	51,578	0/732

Tabel 6.15 Nilai rata-rata Parameter terukur HTB dengan 4 *port*

Nomor *port*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
3001	1	31,867	154,333	52,121	306/732
3002	1	32,000	138,000	83,394	354/732
3003	1	30,067	132,333	62,796	391/732
3004	1	27,033	133,000	19,739	427/732

Tabel 6.16 Nilai rata-rata Parameter terukur CBQ dengan 4 *port*

6.1.3 Skenario Dengan Parameter "Tidak Meminjamkan *Bandwidth Idle*"

Tabel-Tabel selanjutnya merupakan representasi parameter rata-rata yang diukur. Untuk hasil pengujian secara lengkap, dapat dilihat di lembar Lampiran B.

IP *address*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
192.168.1.2	1	69,033	125,000	94,414	0/732
192.168.1.3	1	69,033	125,000	94,157	0/732

Tabel 6.17 Nilai rata-rata Parameter terukur HTB dengan 2 IP *address*

IP *address*	Besar Pesan (MByte)	Waktu pengiriman (detik)	*Throughput* (kbit/s)	*Jitter* (milidetik)	*Loss Datagram*
192.168.1.2	1	26,467	134,667	7,374	429/732
192.168.1.3	1	29,600	135,333	38,831	387/732

Tabel 6.18 Nilai rata-rata Parameter terukur CBQ dengan 2 IP *address*

6.2 Analisis

6.2.1 Skenario Dengan *Leaf Class* Berdasarkan IP *Address*

- ***Throughput***

Dari Tabel 6.1 sampai dengan Tabel 6.8 dapat terlihat dari sisi *Throughput* bisa dipastikan CBQ menghasilkan nilai *Throughput* yang lebih baik dibanding dengan HTB namun karena dilakukan pengaturan *bandwidth* dengan menerapkan suatu *rule* berupa pembatasan *bandwidth* untuk tiap-tiap *client*

maka pada penerapan HTB tidak terjadi kebocoran *bandwidth* dan pada CBQ terjadi kebocoran sehingga dapat dipastikan penerapan manajemen *bandwidth* dengan HTB berdasarkan IP *address* jika dilihat dari parameter *Throughput* lebih baik dibandingkan dengan CBQ.

- **Waktu pemrosesan pesan dalam *bandwidth manager* (t_{bm})**

Waktu pemrosesan pesan dalam CBQ maupun HTB dapat dihitung dengan menggunakan rumus $t_{bm} = t_b - t_m$,

t_{bm} = waktu pemrosesan pesan dalam *bandwidth manager*

t_b = waktu pengiriman pesan melalui *bandwidth manager*

t_m = waktu pengiriman pesan tanpa *bandwidth manager*

$$= \frac{packet\_size \times 8}{line\_speed} \times 1000ms$$

IP *address*	Bandwidth Manager	$t_{b(detik)}$	$t_{m(detik)}$	$t_{bm(detik)}$
192.168.1.2	CBQ	16,667	15,866	0,801
	HTB	17,267	16,439	0,828

Tabel 6.19 Rekapitulasi Perhitungan t_b, t_m, t_{bm} Skenario 1 IP *address*

IP *address*	Bandwidth Manager	$t_{b(detik)}$	$t_{m(detik)}$	$t_{bm(detik)}$
192.168.1.2	CBQ	32,000	31,387	0,613
	HTB	33,667	32,042	1,625
192.168.1.3	CBQ	31,300	31,670	-0,370
	HTB	34,033	32,337	1,696

Tabel 6.20 Rekapitulasi Perhitungan t_b, t_m, t_{bm} Skenario 2 IP *address*

IP *address*	Bandwidth Manager	$t_{b(detik)}$	$t_{m(detik)}$	$t_{bm(detik)}$
192.168.1.2	CBQ	32,367	43,574	-11,207
	HTB	50,167	47,720	2,447
192.168.1.3	CBQ	31,500	46,811	-15,311
	HTB	51,033	48,569	2,464
192.168.1.4	CBQ	29,133	46,545	-17,412
	HTB	51,100	48,665	2,435

Tabel 6.21 Rekapitulasi Perhitungan t_b, t_m, t_{bm} Skenario 3 IP *address*

IP *address*	Bandwidth Manager	$t_{b(detik)}$	$t_{m(detik)}$	$t_{bm(detik)}$
192.168.1.2	CBQ	33,100	54,980	-21,880
	HTB	63,133	60,088	3,045
192.168.1.3	CBQ	31,433	57,555	-26,122
	HTB	64,067	60,832	3,235
192.168.1.4	CBQ	26,467	60,832	-34,365
	HTB	64,033	60,832	3,201
192.168.1.5	CBQ	29,600	60,532	-30,932
	HTB	49,867	65,536	-15,669

Tabel 6.22 Rekapitulasi Perhitungan t_b, t_m, t_{bm} Skenario 4 IP *address*

Dari hasil perhitungan didapat $t_{bm\ CBQ}$ lebih kecil dibanding dengan $t_{bm\ HTB}$, bahkan lebih kecil jika dibandingkan ketika tanpa *bandwidth manager*. Hal ini menandakan untuk pembagian kelas berdasarkan IP *address* jika dilihat dari waktu proses pemrosesan pesan, CBQ lebih cepat dibanding dengan HTB. Namun perlu diingat bahwa pada implementasi *bandwidth manager* menggunakan disiplin antrian CBQ timbul paket *loss* yang sangat besar. *Loss* ini diakibatkan oleh ketidakmampuan CBQ menangani secara simultan layanan dengan *leaf class* yang terlalu banyak.

- ***Jitter***

  Hasil pengukuran memberikan kesimpulan bahwa untuk alokasi *bandwidth* berdasarkan kelas berupa IP *address*, *jitter* pada CBQ lebih baik dibanding dengan HTB, dengan diperolehnya nilai *jitter* yang lebih kecil.

- ***Loss Datagram***

  Loss Datagram CBQ lebih buruk dibanding HTB. *Loss* ini diakibatkan oleh ketidakmampuan CBQ menangani secara simultan layanan dengan *leaf class* yang terlalu banyak. Secara umum *Loss Datagram* yang melebihi toleransi ini menjadikan *Loss Datagram* CBQ pada skenario perbandingan berdasarkan IP *address* lebih buruk dibandingkan HTB.

6.2.2 Skenario Dengan *Leaf Class* Berdasarkan *Port*

- ***Throughput***

  Dari Tabel 6.9 sampai dengan Tabel 6.16 dapat terlihat dari sisi *Throughput* bisa dipastikan CBQ menghasilkan nilai *Throughput* yang lebih baik dibanding dengan HTB namun karena dilakukan pengaturan *bandwidth* dengan menerapkan suatu *rule* berupa pembatasan *bandwidth* untuk tiap-tiap *client* maka pada penerapan HTB tidak terjadi kebocoran *bandwidth* dan pada CBQ terjadi kebocoran sehingga dapat dipastikan penerapan manajemen *bandwidth* dengan HTB berdasarkan IP *address* jika dilihat dari parameter *Throughput* lebih baik dibandingkan dengan CBQ.

- **Waktu pemrosesan pesan dalam *bandwidth manager* (t_{bm})**

IP *address*	Bandwidth Manager	$t_{b(detik)}$	$t_{m(detik)}$	$t_{bm(detik)}$
192.168.1.2	CBQ	16,700	15,876	0,824
	HTB	17,300	16,450	0,850

Tabel 6.23 Rekapitulasi Perhitungan t_b, t_m, t_{bm} Skenario 1 *port*

IP *address*	Bandwidth Manager	$t_{b(detik)}$	$t_{m(detik)}$	$t_{bm(detik)}$
192.168.1.2	CBQ	31,100	31,834	-0,734
	HTB	33,900	32,252	1,648
192.168.1.3	CBQ	32,000	31,307	0,693
	HTB	34,233	32,551	1,682

Tabel 6.24 Rekapitulasi Perhitungan t_b, t_m, t_{bm} Skenario 2 *port*

IP *address*	Bandwidth Manager	$t_{b(detik)}$	$t_{m(detik)}$	$t_{bm(detik)}$
192.168.1.2	CBQ	30,933	43,497	-12,564
	HTB	50,200	47,720	2,480
192.168.1.3	CBQ	31,467	46,634	-15,167
	HTB	51,167	48,762	2,405
192.168.1.4	CBQ	29,700	44,602	-14,902
	HTB	51,033	48,569	2,464

Tabel 6.25 Rekapitulasi Perhitungan t_b, t_m, t_{bm} Skenario 3 *port*

IP *address*	Bandwidth Manager	$t_{b(detik)}$	$t_{m(detik)}$	$t_{bm(detik)}$
192.168.1.2	CBQ	31,867	53,080	-21,213
	HTB	66,067	62,854	3,213
192.168.1.3	CBQ	32,000	59,362	-27,362
	HTB	67,767	64,504	3,263
192.168.1.4	CBQ	30,067	61,904	-31,837
	HTB	67,933	64,674	3,259
192.168.1.5	CBQ	27,033	61,594	-34,561
	HTB	67,733	64,504	3,229

Tabel 6.26 Rekapitulasi Perhitungan t_b, t_m, t_{bm} Skenario 4 *port*

Dari hasil perhitungan didapat $t_{bm\ CBQ}$ lebih kecil dibanding dengan $t_{bm\ HTB}$, bahkan lebih kecil jika dibandingkan ketika tanpa *bandwidth manager*. Hal ini menandakan untuk pembagian kelas berdasarkan *port* jika dilihat dari waktu proses pemrosesan pesan, CBQ lebih cepat dibanding dengan HTB. Namun perlu diingat bahwa pada implementasi *bandwidth manager* menggunakan disiplin antrian CBQ timbul paket *loss* yang sangat besar. *Loss* ini diakibatkan oleh ketidakmampuan CBQ menangani secara simultan layanan dengan *leaf class* yang terlalu banyak.

- ***Jitter***

  Hasil pengukuran memberikan kesimpulan bahwa untuk alokasi *bandwidth* berdasarkan kelas berupa *port*, *jitter* pada CBQ lebih baik dibanding dengan HTB, dengan diperolehnya nilai *jitter* yang lebih kecil.

- ***Loss Datagram***

  Loss Datagram CBQ lebih buruk dibanding HTB. *Loss* ini diakibatkan oleh ketidakmampuan CBQ menangani secara simultan layanan dengan *leaf class* yang terlalu banyak. Secara umum *Loss Datagram* yang melebihi toleransi ini menjadikan *Loss Datagram* CBQ pada skenario perbandingan berdasarkan *port* lebih buruk dibandingkan HTB.

6.2.3 Skenario Dengan Parameter "Tidak Meminjamkan *Bandwidth Idle*"

- ***Throughput***

Tabel 6.17 sampai dengan Tabel 6.18 menunjukkan berfungsinya efek parameter khusus *bounded* untuk CBQ dan *ceil = rate* untuk HTB, saat IP 192.168.1.2 tidak lagi menerima data yang berarti alokasi bandwidthnya tidak sedang dipergunakan (CBQ t > 26,3 detik, HTB t > 69,0 detik), IP 192.168.1.3 yang masih sibuk menerima data tidak dapat menggunakan *bandwidth idle* IP 192.168.1.3 karena terhalang *rule* yang telah dibuat.

Dari Tabel 6.17 sampai dengan Tabel 6.18 juga didapat *Throughput* IP 192.168.1.2 dengan CBQ sebesar 134,667 kbps dan dengan HTB diperoleh 125 kbps, untuk IP 192.168.1.3 sebelum terjadi efek *link sharing* dengan CBQ didapat *Throughput* 135,333 kbps dengan HTB diperoleh 125 kbps. Nampak sekali CBQ mengalami kebocoran penggunaan *bandwidth* tapi tidak untuk HTB, hal ini menandakan HTB lebih baik dibanding CBQ jika dilihat dari kepatuhan terhadap *rule* mengenai *Throughput* yang telah dideklarasikan pada *bandwidth manager*.

- **Waktu pemrosesan pesan dalam Bandwidth Manager (t_{bm})**

IP *address*	Bandwidth Manager	$t_{b(detik)}$	$t_{m(detik)}$	$t_{bm(detik)}$
192.168.1.2	CBQ	26,467	60,832	-34,365
	HTB	69,033	65,536	3,497
192.168.1.3	CBQ	29,600	60,532	-30,932
	HTB	69,033	65,536	3,497

Tabel 6.27 Rekapitulasi Perhitungan t_b, t_m, t_{bm} Skenario 2 IP *address*

Dari hasil perhitungan didapat $t_{bm\ CBQ}$ lebih kecil dibanding dengan $t_{bm\ HTB}$, bahkan lebih kecil jika dibandingkan ketika tanpa *bandwidth manager*. Hal ini menandakan untuk pembagian kelas berdasarkan IP *address* dengan parameter tidak menggunakan *bandwidth idle* jika dilihat dari waktu proses pemrosesan pesan, CBQ lebih cepat dibanding dengan HTB. Namun perlu diingat bahwa pada implementasi *bandwidth manager* menggunakan disiplin antrian CBQ

timbul paket *loss* yang sangat besar. *Loss* ini diakibatkan oleh ketidakmampuan CBQ menangani secara simultan layanan dengan *leaf class* yang terlalu banyak.

- ***Jitter***

  Hasil pengukuran memberikan gambaran bahwa untuk alokasi *bandwidth* berdasarkan kelas berupa IP *address* dengan parameter tidak menggunakan *bandwidth idle*, *jitter* pada CBQ lebih baik dibanding dengan HTB dengan selisih yang cukup signifikan saat *jitter* CBQ mengungguli HTB.

- ***Loss Datagram***

  Loss Datagram CBQ lebih buruk dibanding HTB. *Loss* ini diakibatkan oleh ketidakmampuan CBQ menangani secara simultan layanan dengan *leaf class* yang terlalu banyak. Secara umum *Loss Datagram* yang melebihi toleransi ini menjadikan *Loss Datagram* CBQ pada skenario parameter "tidak meminjamkan *bandwidth idle*" lebih buruk dibandingkan HTB.

BAB VII

PENUTUP

7.1 Kesimpulan

1. *Class Based Queuing* (CBQ) dan *Hierarchical Token Bucket* (HTB) merupakan suatu disiplin antrian yang bersifat *classful*, berarti CBQ dan HTB mempunyai kemampuan untuk menjadwalkan paket berdasarkan kelas-kelas yang tersusun secara hirarki.

2. CBQ dan HTB dapat berfungsi sebagai *bandwidth manager* yang mampu memberikan alokasi *bandwidth* kepada kelas-kelas sesuai dengan kebutuhan dan keinginan pengguna *bandwidth manager*. Kelas-kelas yang dibentuk oleh CBQ maupun HTB dapat berdasarkan pada IP *address* dan nomor *port*.

3. *Throughput* yang dihasilkan kelas yang menggunakan HTB secara umum mampu memenuhi ketentuan/*rule* yang diimplementasikan dibanding CBQ. Hal ini berarti CBQ kurang akurat dibanding HTB dalam hal *throughput* yang dihasilkan pada setiap kelas.

4. Secara umum nilai *jitter* yang diperoleh kelas yang menggunakan CBQ lebih kecil dibanding dengan kelas yang menggunakan HTB.

Skenario	Persentase CBQ terhadap HTB	
	Jitter	Waktu Proses
1 *leaf class* berdasarkan IP Address	47,27	3,37
2 *leaf class* berdasarkan IP Address	67,51	116,49
3 *leaf class* berdasarkan IP Address	82,42	86,02
4 *leaf class* berdasarkan IP Address	85,88	54,40
1 *leaf class* berdasarkan *Port*	47,26	3,16
2 *leaf class* berdasarkan *Port*	62,49	124,52
3 *leaf class* berdasarkan *Port*	65,46	115,86
4 *leaf class* berdasarkan *Port*	61,73	109,30
Tanpa peminjaman bandwidth *idle*	92,17	110,18

Tabel 7.1 Persentase selisih nilai CBQ terhadap HTB

5. Waktu pemrosesan *message*/pesan menggunakan HTB lebih lambat bila dibandingkan menggunakan CBQ, hal ini menandakan datagram yang dikirim oleh CBQ ke *destination*nya lebih cepat tersampaikan dibandingkan menggunakan HTB.

6. Secara umum penggunaan *bandwidth manager* CBQ maupun HTB akan menghasilkan *loss datagram*. CBQ memberikan *loss datagram* sampai 58,60 %, hal ini terjadi pada ketidakmampuan CBQ menangani secara simultan layanan dengan *leaf class* yang terlalu banyak. HTB memberikan nilai yang stabil pada *loss datagram* yaitu nyaris tidak menghasilkan nilai *loss datagram* kecuali pada skenario dengan 4 *leaf class* itupun dengan nilai *loss* yang sangat kecil sekali 5 %, hal ini menandakan secara garis besar *bandwidth manager* HTB lebih baik dalam nilai persentase kehilangan *datagram* dibanding CBQ.

7.2 Saran

1. Pemilihan penggunaan *bandwidth manager* CBQ maupun HTB sebaiknya disesuaikan dengan jenis aplikasi yang sering digunakan. Bila jenis aplikasi mempunyai ukuran data yang kecil dan memerlukan respon layanan yang cepat maka penggunaan CBQ sudah memadai namun bila ukuran datanya besar, membutuhkan keakurasian alokasi *bandwidth* yang tinggi dan tidak mementingkan respon layanan yang cepat maka pilihan HTB adalah lebih baik

2. Untuk simulasi selanjutnya diharapkan menggunakan perangkat yang memadai untuk hasil pengukuran yang lebih baik lagi.

DAFTAR PUSTAKA

[ALM-01]	Almesberger, Werner. 2001. "Linux Networking Traffic Control - Implementation Overview". EPFL ICA.
[ALM-99]	Almesberger, Werner., Salim, Jamal Hadi., dan Kuznetsov, Alexey. 1999. "Differentiated Services on Linux".
[BRE-03]	Brenton, Chriss. 2003. "Network Security Versi Indonesia". Elexmedia Komputindo: Jakarta
[BRO-06]	Brown, Martin A. "Traffic Control HOWTO Version 1.0.2"
[CIS-04]	Cisco Systems Inc. 2004. "Quality of Service Solutions Configuration Guide". http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/qos_c/index.htm. Tanggal akses 30 April 2007
[DEV-02]	Devera, Martin. 2002. "Princip a užití HTB QoS disciplíny". SLT 2002
[DEV-03]	Devera, Martin. 2003. "Hierarchical Token Bucket Theory". http://luxik.cdi.cz/~devik/qos/htb/. Tanggal akses 30 April 2007
[FCI-05]	Anonymous. 2005. "Topology". http://fcit.usf.edu/network/chap5/chap5.htm, Tanggal Akses 30 April 2007
[FER-98]	Ferguson, Paul. 1998. "Quality of Service". John Wiley & Sons: Canada.
[FLO-95]	Floyd, S. dan Jacobson, V. 1995. "Link-sharing and Resource Management Models for Packet Networks". IEEE/ACM Transactions on Networking, Vol. 3 No. 4, hal. 365-386.
[FOR-01]	Forouzan, Behrouz. A. 2001. Data Comunication And Networking, Second Edition. McGraw-Hill. USA
[HOF-94]	Hoffman, D. 1994 "Implementation report on the LBL/UCL/Sun CBQ kernel". Toronto IETF.
[HUB-02]	Hubert, Bert., Maxwell, Gregory., Van Mook, Remco., Van Oosterhout, Martijn., Schroeder, Paul B., Dan Spaans, Jasper. 2002. "Linux Advanced Routing & Traffic Control HOWTO".
[KCH-98]	K. Cho. 1998. "A Framework for Alternate Queueing: Towards Annual Traffic Management by PC-UNIX Based Routers". In Technical Conference, USENIX.
[LAM-04]	Lammle, Todd. 2004. "CCNA Study Guide Versi Indonesia". Elexmedia Komputindo: Jakarta.
[LEO-04]	Leon-Garcia, Alberto. 2004. "Communication Networks : Fundamental Concepts And Key Architectures". McGraw-Hill. USA.

[MIL-05]	Anonymous. 2005. "Networking". http://www.millbury.k12.ma.us/hs/techrepair/networking.html. Tanggal akses : 30 April 2007
[PUR-98]	Purbo, W. Onno. 1998. "Buku Pintar Internet TCP/IP". Elex Media Komputindo: Jakarta.Indonesia
[RAD-99]	Radhakrishnan, Saravanan. 1999. "Linux - Advanced Networking Overview version 1"
[SEM-01]	Semeria, Chuck. 2001. "Supporting Differentiated Service Classes: Queue Scheduling Disciplines". Juniper Networks.
[STE-94]	Stevens, W Richard. 1994. "TCP/IP Illustrated : The Protocols". Addison-Wesley
[TAN-03]	Tanenbaum, Andrew S. 2003. "Computer Networks, Fourth Edition". Prentice Hall.
[TIM-03]	Tim Penelitian Dan Pengembangan Wahana Komputer. 2003. "Konsep Jaringan Komputer Dan Pengembangannya". Wahana Komputer: Jakarta.
[WAK-95]	Wakeman, I., Ghosh, A., Crowcroft, J., Jacobson, V., dan Floyd, S. 1995. "Implementing Real Time Packet Forwarding Policies using Streams". Usenix 1995 Technical Conference. New Orleans, Louisiana, hal. 71-82.

1 LAMPIRAN A

SKRIP HTB.init Dan CBQ.init

1. SKRIP HTB.init

```
#!/bin/bash
#
#    htb.init v0.8.5
#    Copyright (C) 2002-2004  Lubomir Bulej <pallas@kadan.cz>
#
#    chkconfig:   2345 11 89
#    description: script to set up HTB traffic control
#
```

```
#    This program is free software; you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation; either version 2 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program; if not, write to the Free Software
#    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
#    To get the latest version, check on Freshmeat for actual location:
#
#              http://freshmeat.net/projects/htb.init
#
#
# VERSION HISTORY
# ---------------
# v0.8.5- Nathan Shafer <nicodemus at users.sourceforge.net>
#         - allow symlins to class files
#       - Seth J. Blank <antifreeze at users.sourceforge.net>
#         - replace hardcoded ip/tc location with variables
#       - Mark Davis <mark.davis at gmx.de>
#         - allow setting of PRIO_{MARK,RULE,REALM} in class file
# v0.8.4- Lubomir Bulej <pallas at kadan.cz>
#         - fixed small bug in RULE parser to correctly parse
#           rules with identical source and destination fields
#         - removed the experimental INJECT keyword
#         - ignore *~ backup files when looking for classes
#       - Mike Boyer <boyer at administrative.com>
#         - fix to allow arguments to be passed to "restart" command
#       - <face at pos.sk>
#         - fix to preserve class priority after timecheck
# v0.8.3- Lubomir Bulej <pallas at kadan.cz>
#         - use LC_COLLATE="C" when sorting class files
#       - Paulo Sedrez
#         - fix time2abs to allow hours with leading zero in TIME rules
# v0.8.2- Lubomir Bulej <pallas at kadan.cz>
#         - thanks to Hasso Tepper for reporting the following problems
#         - allow dots in interface names for use with VLAN interfaces
#         - fixed a thinko resulting from "cosmetic overdosage" :)
# v0.8.1- Lubomir Bulej <pallas at kadan.cz>
#         - added function alternatives for sed/find with less features. To
#           enable them, you need to set HTB_BASIC to nonempty string.
#         - added posibility to refer to RATE/CEIL of parent class when
#           setting RATE/CEIL for child class. Look for "prate" or "pceil"
#           in the documentation.
#         - fixed broken "timecheck" invocation
# v0.8  - Lubomir Bulej <pallas at kadan.cz>
#         - simplified and converted CBQ.init 0.7 into HTB.init
#         - changed configuration file naming conventions
#         - lots of HTB specific changes
#
#
# INTRODUCTION
# ------------
#
# This script is a clone of CBQ.init and is meant to simplify setup of HTB
# based traffic control. HTB setup itself is pretty simple compared to CBQ,
# so the purpose of this script is to allow the administrator of large HTB
# configurations to manage individual classes using simple, human readable
# files.
#
# The "H" in HTB stands for "hierarchical", so while many people did not use
# (or know about) the possibility to build hierarchical structures using
# CBQ.init, it should be obvious thing to expect from HTB.init :-)
#
# In HTB.init this is done differently, compared to CBQ.init: the usage of
# PARENT keyword was dropped and instead, class file naming convetion was
# introduced. This convention allows the child class to determine ID of its
# parent class from the filename and also (if not abused :) enforces file
# ordering so that the parent classes are created before their children.
#
# HTB.init uses simple caching mechanism to speed up "start" invocation if the
# configuration is unchanged. When invoked for the first time, it compiles the
# configuration files into simple shell script containing the sequence of "tc"
# commands required to setup the traffic control. This cache-script is stored
# in /var/cache/htb.init by default and is invalidated either by presence of
# younger class config file, or by invoking HTB.init with "start invalidate".
#
```

```
# If you want to HTB.init to setup the traffic control directly without the
# cache, invoke it with "start nocache" parameters. Caching is also disabled
# if you have logging enabled (ie. HTB_DEBUG is not empty).
#
# If you only want HTB.init to translate your configuration to "tc" commands,
# invoke it using the "compile" command. Bear in mind that "compile" does not
# check if the "tc" commands were successful - this is done (in certain places)
# only when invoked with "start nocache" command. When you are testing your
# configuration, you should use it to check whether it is completely valid.
#
# In case you are getting strange sed/find errors, try to uncomment line with
# HTB_BASIC setting, or set the variable to nonempty string. This will enable
# function alternatives which require less advanced sed/find functionality. As
# a result, the script will run slower but will probably run. Also the caching
# will not work as expected and you will have to invalidate the cache manually
# by invoking HTB.init with "start invalidate".
#
#
# CONFIGURATION
# -------------
#
# Every traffic class is described by a single file in placed in $HTB_PATH
# directory, /etc/sysconfig/htb by default. The naming convention is different
# compared to CBQ.init. First notable change is missing 'htb-' prefix. This
# was replaced by interface name to improve human readability and to separate
# qdisc-only configuration.
#
# Global qdisc options are placed in $HTB_PATH/<ifname>, where <ifname> is
# (surprisingly) name of the interface, made of characters and numbers. This
# file must be present if you want to setup HTB on that interface. If you
# don't have any options to put into it, leave it empty, but present.
#
# Class options belong to files with names matching this expression:
# $HTB_PATH/<ifname>-<clsid>(:<clsid>)*<description>
#
# <clsid> is class ID which is hexadecimal number in range 0x2-0xFFFF, without
# the "0x" prefix. If a colon-delimited list of class IDs is specified, the
# last <clsid> in the list represents ID of the class in the config file.
#
# <clsid> preceding the last <clsid> is class ID of the parent class. To keep
# ordering so that parent classes are always created before their children, it
# is recommended to include full <clsid> path from root class to the leaf one.
#
# <description> is (almost) arbitrary string where you can put symbolic
# class names for better readability.
#
# Examples of valid names:
#
#       eth0-2          root class with ID 2, on device eth0
#       eth0-2:3        child class with ID 3 and parent 2, on device eth0
#       eth0-2:3:4      child class with ID 4 and parent 3, on device eth0
#       eth1-2.root     root class with ID 2, on device eth1
#
#
# The configuration files may contain the following parameters. For detailed
# description of HTB parameters see http://luxik.cdi.cz/~devik/qos/htb.
#
### HTB qdisc parameters
#
# The following parameters apply to HTB root queuening discipline only and
# are expected to be put into $HTB_PATH/<ifname> files. These files must
# exist (even empty) if you want to configure HTB on given interface.
#
# DEFAULT=<clsid>                               optional, default 0
# DEFAULT=30
#
#       <dclsid> is ID of the default class where UNCLASSIFIED traffic goes.
#       Unlike HTB qdisc, HTB.init uses 0 as default class ID, which is
#       internal FIFO queue that will pass packets along at FULL speed!
#
#       If you want to avoid surprises, always define default class and
#       allocate minimal portion of bandwidth to it.
#
# R2Q=<number>                                  optional, default 10
# R2Q=100
#
#       This allows you to set coefficient for computing DRR (Deficit
#       Round Robin) quanta. The default value of 10 is good for rates
#       from 5-500kbps and should be increased for higher rates.
#
# DCACHE=yes|no                                         optional, default "no"
#
#       This parameters turns on "dequeue cache" which results in degraded
```

```
#       fairness but allows HTB to be used on very fast network devices.
#       This is turned off by default.
#
### HTB class parameters
#
# The following are parameters for HTB classes and are expected
# to be put into $HTB_PATH/<ifname>-<clsid>(:<clsid>)*.* files.
#
# RATE=<speed>|prate|pceil                    mandatory
# RATE=5Mbit
#
#       Bandwidth allocated to the class. Traffic going through the class is
#       shaped to conform to specified rate. You can use Kbit, Mbit or bps,
#       Kbps and Mbps as suffices. If you don't specify any unit, bits/sec
#       are used. Also note that "bps" means "bytes per second", not bits.
#
#       The "prate" or "pceil" values will resolve to RATE or CEIL of parent
#       class. This feature is meant to help humans to keep configuration
#       files consistent.
#
# CEIL=<speed>|prate|pceil                    optional, default $RATE
# CEIL=6MBit
#
#       The maximum bandwidth that can be used by the class. The difference
#       between CEIL and RATE amounts to bandwidth the class can borrow, if
#       there is unused bandwidth left.
#
#       By default, CEIL is equal to RATE so the class cannot borrow bandwidth
#       from its parent. If you want the class to borrow unused bandwidth, you
#       must specify the maximal amount it can use, if available.
#
#       When several classes compete for the unused bandwidth, each of the
#       classes is given share proportional to their RATE.
#
# BURST=<bytes>                               optional, default computed
# BURST=10Kb
#
# CBURST=<bytes>                              optional, default computed
# CBURST=2Kb
#
#       BURST and CBURST parameters control the amount of data that can
#       be sent from one class at maximum (hardware) speed before trying
#       to service other class.
#
#       If CBURST is small (one packet size) it shapes bursts not to
#       exceed CEIL rate the same way PEAK works for TBF.
#
# PRIO=<number>                               optional, default 0
# PRIO=5
#
#       Priority of class traffic. The higher the number, the lesser the
#       priority. Also, classes with higher priority are offered excess
#       bandwidth first.
#
# LEAF=none|sfq|pfifo|bfifo                   optional, default "none"
#
#       Tells the script to attach specified leaf queueing discipline to HTB
#       class. By default, no leaf qdisc is used.
#
#       If you want to ensure (approximately) fair sharing of bandwidth among
#       several hosts in the same class, you should specify LEAF=sfq to attach
#       SFQ as leaf queueing discipline to the class.
#
# MTU=<bytes>                                 optional, default "1600"
#
#       Maximum packet size HTB creates rate maps for. The default should
#       be sufficient for most cases, it certainly is for Ethernet.
#
### SFQ qdisc parameters
#
# The SFQ queueing discipline is a cheap way to fairly share class bandwidth
# among several hosts. The fairness is approximate because it is stochastic,
# but is not CPU intensive and will do the job in most cases. If you desire
# real fairness, you should probably use WRR (weighted round robin) or WFQ
# queueing disciplines. Note that SFQ does not do any traffic shaping - the
# shaping is done by the HTB class the SFQ is attached to.
#
# QUANTUM=<bytes>                             optional, qdisc default
#
#       Amount of data in bytes a stream is allowed to dequeue before next
#       queue gets a turn. Defaults to one MTU-sized packet. Do not set
#       this parameter below the MTU!
#
```

```
# PERTURB=<seconds>                             optional, default "10"
#
#       Period of hash function perturbation. If unset, hash reconfiguration
#       will never take place which is what you probably don't want. The
#       default value of 10 seconds is probably a good value.
#
### PFIFO/BFIFO qdisc parameters
#
# Those are simple FIFO queueing disciplines. They only have one parameter
# which determines their length in bytes or packets.
#
# LIMIT=<packets>|<bytes>                       optional, qdisc default
# LIMIT=1000
#
#       Number of packets/bytes the queue can hold. The unit depends on
#       the type of queue used.
#
### Filtering parameters
#
# RULE=[[saddr[/prefix]][:port[/mask]],][daddr[/prefix]][:port[/mask]]
#
#       These parameters make up "u32" filter rules that select traffic for
#       each of the classes. You can use multiple RULE fields per config.
#
#       The optional port mask should only be used by advanced users who
#       understand how the u32 filter works.
#
# Some examples:
#
#       RULE=10.1.1.0/24:80
#              selects traffic going to port 80 in network 10.1.1.0
#
#       RULE=10.2.2.5
#              selects traffic going to any port on single host 10.2.2.5
#
#       RULE=10.2.2.5:20/0xfffe
#              selects traffic going to ports 20 and 21 on host 10.2.2.5
#
#       RULE=:25,10.2.2.128/26:5000
#              selects traffic going from anywhere on port 50 to
#              port 5000 in network 10.2.2.128
#
#       RULE=10.5.5.5:80,
#              selects traffic going from port 80 of single host 10.5.5.5
#
#
#
# REALM=[srealm,][drealm]
#
#       These parameters make up "route" filter rules that classify traffic
#       according to packet source/destination realms. For information about
#       realms, see Alexey Kuznetsov's IP Command Reference. This script
#       does not define any realms, it justs builds "tc filter" commands
#       for you if you need to classify traffic this way.
#
#       Realm is either a decimal number or a string referencing entry in
#       /etc/iproute2/rt_realms (usually).
#
# Some examples:
#
#       REALM=russia,internet
#              selects traffic going from realm "russia" to realm "internet"
#
#       REALM=freenet,
#              selects traffic going from realm "freenet"
#
#       REALM=10
#              selects traffic going to realm 10
#
#
#
# MARK=<mark>
#
#       These parameters make up "fw" filter rules that select traffic for
#       each of the classes accoring to firewall "mark". Mark is a decimal
#       number packets are tagged with if firewall rules say so. You can
#       use multiple MARK fields per config.
#
#
# Note: Rules for different filter types can be combined. Attention must be
#       paid to the priority of filter rules, which can be set below through
#       the PRIO_{RULE,MARK,REALM} variables.
#
```

```
### Time ranging parameters
#
# TIME=[<dow><dow>.../]<from>-<till>;<rate>[/<burst>][,<ceil>[/<cburst>]]
# TIME=60123/18:00-06:00;256Kbit/10Kb,384Kbit
# TIME=18:00-06:00;256Kbit
#
#       This parameter allows you to change class bandwidth during the day or
#       week. You can use multiple TIME rules. If there are several rules with
#       overlapping time periods, the last match is taken. The <rate>, <burst>,
#       <ceil> and <cburst> fields correspond to parameters RATE, BURST, CEIL
#       and CBURST.
#
#       <dow> is single digit in range 0-6 and represents day of week as
#       returned by date(1). To specify several days, just concatenate the
#       digits together.
#
#
#
# TRIVIAL EXAMPLE
# ---------------
#
# Consider the following example:
# (taken from Linux Advanced Routing & Traffic Control HOWTO)
#
# You have a Linux server with total of 5Mbit available bandwidth. On this
# machine, you want to limit webserver traffic to 5Mbit, SMTP traffic to 3Mbit
# and everything else (unclassified traffic) to 1Kbit. In case there is unused
# bandwidth, you want to share it between SMTP and unclassified traffic.
#
# The "total bandwidth" implies one top-level class with maximum bandwidth
# of 5Mbit. Under the top-level class, there are three child classes.
#
# First, the class for webserver traffic is allowed to use 5Mbit of bandwidth.
#
# Second, the class for SMTP traffic is allowed to use 3Mbit of bandwidth and
# if there is unused bandwidth left, it can use it but must not exceed 5Mbit
# in total.
#
# And finally third, the class for unclassified traffic is allowed to use
# 1Kbit of bandwidth and borrow unused bandwith, but must not exceed 5Mbit.
#
# If there is demand in all classes, each of them gets share of bandwidth
# proportional to its default rate. If there unused is bandwidth left, they
# (again) get share proportional to their default rate.
#
# Configuration files for this scenario:
# ------------------------------------------------------------------------
# eth0          eth0-2.root    eth0-2:10.www  eth0-2:20.smtp eth0-2:30.dfl
# ----          -----------    -------------  -------------- -------------
# DEFAULT=30    RATE=5Mbit     RATE=5Mbit     RATE=3Mbit     RATE=1Kbit
#               BURST=15k      BURST=15k      CEIL=5Mbit     CEIL=5Mbit
#                              LEAF=sfq       BURST=15k      BURST=15k
#                              RULE=*:80,     LEAF=sfq       LEAF=sfq
#                                             RULE=*:25
# ------------------------------------------------------------------------
#
# Remember that you can only control traffic going out of your linux machine.
# If you have a host connected to network and want to control its traffic on
# the gateway in both directions (with respect to the host), you need to setup
# traffic control for that host on both (or all) gateway interfaces.
#
# Enjoy.
#
###############################################################################

export LC_ALL=C

### Command locations
TC=/sbin/tc
IP=/sbin/ip
MP=/sbin/modprobe

### Default filter priorities (must be different)
PRIO_RULE_DEFAULT=${PRIO_RULE:-100}
PRIO_MARK_DEFAULT=${PRIO_MARK:-200}
PRIO_REALM_DEFAULT=${PRIO_REALM:-300}

### Default HTB_PATH & HTB_CACHE settings
HTB_PATH=${HTB_PATH:-/etc/htb}
HTB_CACHE=${HTB_CACHE:-/var/cache/htb.init}

### Uncomment for sed/find with less features (useful for busybox)
#HTB_BASIC="yes"
```

```
### Uncomment to enable logfile for debugging
#HTB_DEBUG="/var/run/htb-$1"

### Modules to probe for. Uncomment the last HTB_PROBE
### line if you have QoS support compiled into kernel
HTB_PROBE="sch_htb sch_sfq cls_fw cls_u32 cls_route"
#HTB_PROBE=""

### Config keywords
HTB_QDISC="DEFAULT\|DCACHE\|R2Q"
HTB_CLASS="RATE\|CEIL\|BURST\|CBURST\|PRIO\|LEAF\|MTU"
HTB_CLASS="$HTB_CLASS\|PRIO_RULE\|PRIO_MARK\|PRIO_REALM"
HTB_CLASS="$HTB_CLASS\|LIMIT\|QUANTUM\|PERTURB"


##############################################################################
############################ SUPPORT FUNCTIONS ###############################
##############################################################################

if [ -z "$HTB_BASIC" ]; then
        ### List of network devices
        all_device_list () {
                ip link show \
                | sed -n "/^[0-9]/ { s/[[:space:]]//g; \
                s/^[0-9]\+:\([^@-]\+\)\(@.\+\)\?:<.*/\1/; p; }"
        } # all_device_list


        ### Load & filter file $HTB_PATH/$1
        htb_filter_file () {
                sed -n "s/#.*//; s/[^a-zA-Z0-9.,;:=/*-_]\+//g; \
                /^[a-zA-Z0-9]\+=[a-zA-Z0-9.,:;/*-_]\+$/ p" $HTB_PATH/$1
        } # htb_filter_file


        ### Parse class ID chain from file name
        htb_clsid_chain () {
                echo "${1#*-}" \
                | sed -n "/^[0-9a-fA-F]/ { s/^\([0-9a-fA-F:]\+\).*/\1/; \
                s/::/:/g; s/:$//; p; }"
        } # htb_clsid_chain


        ### List of classes in $HTB_PATH
        htb_class_list () {
                for dev in `htb_device_list`; do
                        find $HTB_PATH \( -type f -or -type l \) \
                        -name "$dev-*" -not -name '*~' -maxdepth 1 \
                        -printf "%f\n"| sort
                done
        } # htb_class_list

        ### Gather $1 rules from $CFILE
        htb_cfile_rules () {
                echo "$CFILE"| sed -n "/^$1=/ { s/.*=//; p; }"
        } # htb_cfile_rules


        ### Validate cache against config files
        htb_valid_cache () {
                for dev in `htb_device_list`; do
                        [ `find $HTB_PATH \( -type f -or -type l \) \
                          -name "$dev*" -maxdepth 1 -newer $HTB_CACHE| \
                          wc -l` -gt 0 ] && VALID=0
                        [ $VALID -ne 1 ] && break
                done
        } # htb_valid_cache


        ### Find class config for device $1, which is newer than cache
        htb_cache_older () {
                [ `find $HTB_PATH -type f -name "$1*" -maxdepth 1 \
                   -newer $HTB_CACHE| wc -l` -gt 0 ] && return 0
                return 1
        } # htb_cache_older


        ### Get current RATE and CEIL
        htb_class_state () {
                tc class show dev $1 \
                | sed -n "s/[[:space:]]\+/ /g; /^class htb 1:$2 / \
                { s/.*rate \(.\+\) burst.*/\1/; p; q; }"
```

```
        } # htb_class_state

else ### Less feature-hungry versions of above functions

        all_device_list () {
                ip link show \
                | grep "^[0-9]" \
                | sed "s/[[:space:]]//g; \
                s/^[0-9]\+:\([^@-]\+\)\(@.\+\)\?:<.*/\1/"
        } # all_device_list

        htb_filter_file () {
                sed 's/#.*//; s/[^a-zA-Z0-9.,:=/*-_]\+//g' $HTB_PATH/$1 \
                | grep '^[a-zA-Z0-9]\+=[a-zA-Z0-9.,:/*-_]\+$'
        } # htb_filter_file

        htb_clsid_chain () {
                echo "${1#*-}" \
                | grep '^[a-fA-F0-9]' \
                | sed 's/^\([a-fA-F0-9:]\+\).*/\1/; s/::/:/g; s/:$//'
        } # htb_clsid_chain

        htb_class_list () {
                PFX=`echo "$HTB_PATH"| sed 's/\//\\\\\//g'`
                for dev in `htb_device_list`; do
                        find $HTB_PATH -type f -name "$dev-*" \
                        | grep "^$HTB_PATH/$dev-[^/]\+[^~]$" \
                        | sed "s/$PFX\///" \
                        | sort
                done
        } # htb_class_list

        htb_cfile_rules () {
                echo "$CFILE"| grep "^$1="| cut -d"=" -f2
        } # htb_cfile_rules

        htb_cache_older () {
                ### cache is always up-to-date
                return 1
        } # htb_cache_older

        htb_class_state () {
                tc class show dev $1 \
                | sed 's/[[:space:]]\+/ /g' \
                | grep "^class htb 1:$2 " \
                | sed 's/.*rate \(.\+\) burst.*/\1/'
        } # htb_class_state
fi # HTB_BASIC


### List of HTB devices
htb_device_list () {
        for dev in `all_device_list`; do
                [ -f $HTB_PATH/$dev ] && echo $dev
        done
} # htb_device_list


### Remove root class from device $1
htb_device_off () {
        tc qdisc del dev $1 root 2> /dev/null
} # htb_device_off


### Remove HTB from all devices
htb_off () {
        for dev in `htb_device_list`; do
                htb_device_off $dev
        done
} # htb_off


### Prefixed message
htb_message () {
        echo -e "**HTB: $@"
} # htb_message

### Failure message
htb_failure () {
        htb_message "$@"
        exit 1
} # htb_failure
```

```
### Failure w/htb_off
htb_fail_off () {
        htb_message "$@"
        htb_off
        exit 1
} # htb_fail_off


### Convert time to absolute value
htb_time2abs () {
        local min=${1##*:}; min=${min##0}
        local hrs=${1%%:*}; hrs=${hrs##0}
        echo $[hrs*60 + min]
} # htb_time2abs


### Display traffic control setup
htb_show () {
        for dev in `all_device_list`; do
                [ `tc qdisc show dev $dev| wc -l` -eq 0 ] && continue
                echo -e "### $dev: queueing disciplines\n"
                tc $1 qdisc show dev $dev; echo

                [ `tc class show dev $dev| wc -l` -eq 0 ] && continue
                echo -e "### $dev: traffic classes\n"
                tc $1 class show dev $dev; echo

                [ `tc filter show dev $dev| wc -l` -eq 0 ] && continue
                echo -e "### $dev: filtering rules\n"
                tc $1 filter show dev $dev; echo
        done
} # htb_show



### Derive DEVICE, CLASS and PARENT from $1
### Check validity of CLASS and PARENT class IDs
### Load class configuration from $HTP_PATH/$1
### Configure class parameters from CFILE
htb_load_class () {
        DEVICE=${1%%-*}
        CLSIDS=`htb_clsid_chain $1`
        CLASS=${CLSIDS##*:}; [ -z "$CLASS" ] &&
                htb_fail_off "$1 has invalid class ID!"

        [ $[0x$CLASS] -lt 2 -o $[0x$CLASS] -gt 65535 ] &&
                htb_fail_off "class ID of $1 must be in range 0x2-0xFFFF!"

        CLSIDS=${CLSIDS%$CLASS}; CLSIDS=${CLSIDS%:}
        PARENT=${CLSIDS##*:}; [ -n "$PARENT" ] &&
                [ $[0x$PARENT] -lt 2 -o $[0x$PARENT] -gt 65535 ] &&
                        htb_fail_off "parent ID of $1 must be in range 0x2-0xFFFF!"

        CFILE=`htb_filter_file $1`


        ### Set defaults & load class
        MTU=""; LEAF=none; PERTURB=10
        RATE=""; BURST=""; CEIL=""; CBURST=""
        PRIO=""; LIMIT=""; QUANTUM=""

        PRIO_RULE=$PRIO_RULE_DEFAULT
        PRIO_MARK=$PRIO_MARK_DEFAULT
        PRIO_REALM=$PRIO_REALM_DEFAULT

        eval `echo "$CFILE"| grep "^\($HTB_CLASS\)="`
        RNAME=""; CNAME=""

        ### Resolve RATE if needed
        [ "$RATE" = "prate" ] && RNAME=RATE_$PARENT
        [ "$RATE" = "pceil" ] && RNAME=CEIL_$PARENT
        [ -n "$RNAME" ] && RATE=${!RNAME}

        ### RATE is required
        [ -z "$RATE" ] &&
                htb_fail_off "missing or unresolvable RATE in $1!"

        ### Resolve CEIL if needed
        [ "$CEIL" = "prate" ] && CNAME=RATE_$PARENT
        [ "$CEIL" = "pceil" ] && CNAME=CEIL_$PARENT
        [ -n "$CNAME" ] && CEIL=${!CNAME}

        ### Store CEIL & RATE for children
```

```
        eval RATE_$CLASS=$RATE
        eval CEIL_$CLASS=${CEIL:-$RATE}
} # htb_load_class


################################################################################
################################### INIT #######################################
################################################################################

### Check iproute2 tools
[ -x $TC -a -x $IP ] ||
        htb_failure "iproute2 utilities not installed or executable!"

### Check $HTB_PATH directory
[ -d $HTB_PATH -a -r $HTB_PATH -a -x $HTB_PATH ] ||
        htb_failure "$HTB_PATH does not exist or is not readable!"

### ip/tc wrappers
if [ "$1" = "compile" ]; then
        ### no module probing
        HTB_PROBE=""

        ip () {
                $IP "$@"
        } # ip

        ### echo-only version of "tc" command
        tc () {
                echo "$TC $@"
        } # tc

elif [ -n "$HTB_DEBUG" ]; then
        echo -e "# `date`" > $HTB_DEBUG

        ### Logging version of "ip" command
        ip () {
                echo -e "\n# ip $@" >> $HTB_DEBUG
                $IP "$@" 2>&1 | tee -a $HTB_DEBUG
        } # ip

        ### Logging version of "tc" command
        tc () {
                echo -e "\n# tc $@" >> $HTB_DEBUG
                $TC "$@" 2>&1 | tee -a $HTB_DEBUG
        } # tc
else
        # default wrappers

        ip () {
                $IP "$@"
        } # ip

        tc () {
                $TC "$@"
        } # tc
fi # ip/tc wrappers


case "$1" in

################################################################################
############################### START/COMPILE ##################################
################################################################################

start|compile)

### Probe QoS modules (start only)
for module in $HTB_PROBE; do
        $MP $module || htb_failure "failed to load module $module"
done

### If we are in compile/nocache/logging mode, don't bother with cache
if [ "$1" != "compile" -a "$2" != "nocache" -a -z "$HTB_DEBUG" ]; then
        VALID=1

        ### validate the cache
        [ "$2" = "invalidate" -o ! -f $HTB_CACHE ] && VALID=0
        [ $VALID -eq 1 ] && for dev in `htb_device_list`; do
                htb_cache_older $dev && VALID=0
                [ $VALID -ne 1 ] && break
        done

        ### compile the config if the cache is invalid
```

```
        if [ $VALID -ne 1 ]; then
                $0 compile > $HTB_CACHE ||
                        htb_fail_off "failed to compile HTB configuration!"
        fi

        ### run the cached commands
        exec /bin/sh $HTB_CACHE 2> /dev/null
fi


### Setup root qdisc on all configured devices
DEVICES=`htb_device_list`
[ -z "$DEVICES" ] && htb_failure "no configured devices found!"

for dev in $DEVICES; do
        ### Retrieve root qdisc options
        DEFAULT=""; DCACHE=""; R2Q=""
        eval `htb_filter_file $dev| grep "^\($HTB_QDISC\)="`
        [ "$DCACHE" = "yes" ] && DCACHE="dcache" || DCACHE=""

        ### Remove old root qdisc from device
        htb_device_off $dev

        ### Setup root qdisc for the device
        tc qdisc add dev $dev root handle 1 htb \
        default ${DEFAULT:-0} ${R2Q:+r2q $R2Q} $DCACHE ||
                htb_fail_off "failed to set root qdisc on $dev!"

        [ "$1" = "compile" ] && echo
done # dev


### Setup traffic classes (if configured)
for classfile in `htb_class_list`; do
        htb_load_class $classfile

        ### Create the class
        tc class add dev $DEVICE parent 1:$PARENT classid 1:$CLASS \
        htb rate $RATE ${CEIL:+ceil $CEIL} ${BURST:+burst $BURST} \
        ${PRIO:+prio $PRIO} ${CBURST:+cburst $CBURST} ${MTU:+mtu $MTU} ||
                htb_fail_off "failed to add class $CLASS with parent $PARENT on $DEVICE!"

        ### Create leaf qdisc if set
        if [ "$LEAF" != "none" ]; then
                if [ "$LEAF" = "sfq" ]; then
                        LEAFPARM="${PERTURB:+perturb $PERTURB} ${QUANTUM:+quantum
$QUANTUM}"
                elif [ "$LEAF" = "pfifo" -o "$LEAF" = "bfifo" ]; then
                        LEAFPARM="${LIMIT:+limit $LIMIT}"
                else
                        htb_fail_off "unknown leaf qdisc ($LEAF) in $classfile!"
                fi

                tc qdisc add dev $DEVICE \
                parent 1:$CLASS handle $CLASS $LEAF $LEAFPARM ||
                        htb_fail_off "failed to add leaf qdisc to class $CLASS on
$DEVICE!"
        fi


        ### Create fw filter for MARK fields
        for mark in `htb_cfile_rules MARK`; do
                ### Attach fw filter to root class
                tc filter add dev $DEVICE parent 1:0 protocol ip \
                prio $PRIO_MARK handle $mark fw classid 1:$CLASS
        done ### mark

        ### Create route filter for REALM fields
        for realm in `htb_cfile_rules REALM`; do
                ### Split realm into source & destination realms
                SREALM=${realm%%,*}; DREALM=${realm##*,}
                [ "$SREALM" = "$DREALM" ] && SREALM=""

                ### Convert asterisks to empty strings
                SREALM=${SREALM#\*}; DREALM=${DREALM#\*}

                ### Attach route filter to the root class
                tc filter add dev $DEVICE parent 1:0 protocol ip \
                prio $PRIO_REALM route ${SREALM:+from $SREALM} \
                ${DREALM:+to $DREALM} classid 1:$CLASS
        done ### realm

        ### Create u32 filter for RULE fields
```

```
        for rule in `htb_cfile_rules RULE`; do
                ### Split rule into source & destination
                SRC=${rule%%,*}; DST=${rule##*,}
                [ "$SRC" = "$rule" ] && SRC=""


                ### Split destination into address, port & mask fields
                DADDR=${DST%%:*}; DTEMP=${DST##*:}
                [ "$DADDR" = "$DST" ] && DTEMP=""

                DPORT=${DTEMP%%/*}; DMASK=${DTEMP##*/}
                [ "$DPORT" = "$DTEMP" ] && DMASK="0xffff"


                ### Split up source (if specified)
                SADDR=""; SPORT=""
                if [ -n "$SRC" ]; then
                        SADDR=${SRC%%:*}; STEMP=${SRC##*:}
                        [ "$SADDR" = "$SRC" ] && STEMP=""

                        SPORT=${STEMP%%/*}; SMASK=${STEMP##*/}
                        [ "$SPORT" = "$STEMP" ] && SMASK="0xffff"
                fi


                ### Convert asterisks to empty strings
                SADDR=${SADDR#\*}; DADDR=${DADDR#\*}

                ### Compose u32 filter rules
                u32_s="${SPORT:+match ip sport $SPORT $SMASK}"
                u32_s="${SADDR:+match ip src $SADDR} $u32_s"
                u32_d="${DPORT:+match ip dport $DPORT $DMASK}"
                u32_d="${DADDR:+match ip dst $DADDR} $u32_d"

                ### Uncomment the following if you want to see parsed rules
                #echo "$rule: $u32_s $u32_d"

                ### Attach u32 filter to the appropriate class
                tc filter add dev $DEVICE parent 1:0 protocol ip \
                prio $PRIO_RULE u32 $u32_s $u32_d classid 1:$CLASS
        done ### rule

        [ "$1" = "compile" ] && echo
done ### classfile
;;


################################################################################
################################## TIME CHECK ##################################
################################################################################

timecheck)

### Get time + weekday
TIME_TMP=`date +%w/%k:%M`
TIME_DOW=${TIME_TMP%%/*}
TIME_NOW=${TIME_TMP##*/}
TIME_ABS=`htb_time2abs $TIME_NOW`

### Check all classes (if configured)
for classfile in `htb_class_list`; do
        ### Load class and gather all TIME rules
        htb_load_class $classfile
        TIMESET=`htb_cfile_rules TIME`
        [ -z "$TIMESET" ] && continue

        MATCH=0; CHANGE=0
        for timerule in $TIMESET; do
                ### Split TIME rule to pieces
                TIMESPEC=${timerule%%;*}; PARAMS=${timerule##*;}
                WEEKDAYS=${TIMESPEC%%/*}; INTERVAL=${TIMESPEC##*/}
                BEG_TIME=${INTERVAL%%-*}; END_TIME=${INTERVAL##*-}

                ### Check the day-of-week (if present)
                [ "$WEEKDAYS" != "$INTERVAL" -a \
                  -n "${WEEKDAYS##*$TIME_DOW*}" ] && continue

                ### Compute interval boundaries
                BEG_ABS=`htb_time2abs $BEG_TIME`
                END_ABS=`htb_time2abs $END_TIME`

                ### Midnight wrap fixup
                if [ $BEG_ABS -gt $END_ABS ]; then
```

```
                        [ $TIME_ABS -le $END_ABS ] &&
                                TIME_ABS=$[TIME_ABS + 24*60]

                        END_ABS=$[END_ABS + 24*60]
                fi

                ### If time period matches, remember params and set MATCH flag
                if [ $TIME_ABS -ge $BEG_ABS -a $TIME_ABS -lt $END_ABS ]; then
                        RATESPEC=${PARAMS%%,*}; CEILSPEC=${PARAMS##*,}
                        [ "$RATESPEC" = "$CEILSPEC" ] && CEILSPEC=""

                        NEW_RATE=${RATESPEC%%/*}; NEW_BURST=${RATESPEC##*/}
                        [ "$NEW_RATE" = "$NEW_BURST" ] && NEW_BURST=""

                        NEW_CEIL=${CEILSPEC%%/*}; NEW_CBURST=${CEILSPEC##*/}
                        [ "$NEW_CEIL" = "$NEW_CBURST" ] && NEW_CBURST=""

                        MATCH=1
                fi
        done ### timerule


        ### Get current RATE and CEIL of a class
        read RATE_NOW JUNK CEIL_NOW <<-EOT
        `htb_class_state $DEVICE $CLASS`
        EOT

        [ -z "$RATE_NOW" -o -z "$CEIL_NOW" ] && continue


        ### Fill empty values if matched
        if [ $MATCH -ne 0 ]; then
                NEW_RATE=${NEW_RATE:-$RATE_NOW}
                NEW_CEIL=${NEW_CEIL:-$CEIL_NOW}

                NEW_BURST=${NEW_BURST:-$BURST}
                NEW_CBURST=${NEW_CBURST:-$CBURST}

        ### Force configured values if not matched
        else
                NEW_RATE=$RATE; NEW_CEIL=$CEIL
                NEW_BURST=$BURST; NEW_CBURST=$CBURST
        fi



        ### Check for RATE and CEIL changes
        [ "$RATE_NOW" != "$NEW_RATE" ] && CHANGE=1
        [ "$CEIL_NOW" != "$NEW_CEIL" ] && CHANGE=1

        ### If there are no changes, go for next class
        [ $CHANGE -eq 0 ] && continue


        ### Replace HTB class
        tc class change dev $DEVICE classid 1:$CLASS htb \
        prio $PRIO rate $NEW_RATE ${NEW_CEIL:+ceil $NEW_CEIL} \
        ${NEW_BURST:+burst $NEW_BURST} ${NEW_CBURST:+cburst $NEW_CBURST}

        htb_message "$TIME_NOW: change on $DEVICE:$CLASS ($RATE_NOW/$CEIL_NOW ->
$NEW_RATE/$NEW_CEIL)"
done ### class file
;;


################################################################################
################################### THE REST ###################################
################################################################################

stop)
        htb_off
        ;;

list)
        htb_show
        ;;

stats)
        htb_show -s
        ;;

restart)
        shift
```

```
        $0 stop
        $0 start "$@"
        ;;

*)
        echo "Usage: `basename $0` {start|compile|stop|restart|timecheck|list|stats}"
esac
```

2. SKRIP CBQ.init

```
#!/bin/bash
#
#    cbq.init v0.7.3
#    Copyright (C) 1999  Pavel Golubev <pg@ksi-linux.com>
#    Copyright (C) 2001-2004  Lubomir Bulej <pallas@kadan.cz>
#
#    chkconfig:   2345 11 89
#    description: sets up CBQ-based traffic control
#
#    This program is free software; you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation; either version 2 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program; if not, write to the Free Software
#    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
#    To get the latest version, check on Freshmeat for actual location:
#
#              http://freshmeat.net/projects/cbq.init
#
#
# VERSION HISTORY
# ---------------
# v0.7.3- Deepak Singhal <singhal at users.sourceforge.net>
#         - fix timecheck to not ignore regular TIME rules after
#           encountering a TIME rule that spans over midnight
#       - Nathan Shafer <nicodemus at users.sourceforge.net>
#         - allow symlinks to class files
#       - Seth J. Blank <antifreeze at users.sourceforge.net>
#         - replace hardcoded ip/tc location with variables
#       - Mark Davis <mark.davis at gmx.de>
#         - allow setting of PRIO_{MARK,RULE,REALM} in class file
#       - Fernando Sanch <toptnc at users.sourceforge.net>
#         - allow underscores in interface names
# v0.7.2- Paulo Sedrez
#         - fix time2abs to allow hours with leading zero in TIME rules
#       - Svetlin Simeonov <zvero at yahoo.com>
#         - fix cbq_device_list to allow VLAN interfaces
#       - Mark Davis <mark.davis at gmx.de>
#         - ignore *~ backup files when looking for classes
#       - Mike Boyer <boyer at administrative.com>
#         - fix to allow arguments to be passed to "restart" command
# v0.7.1- Lubomir Bulej <pallas at kadan.cz>
#         - default value for PERTURB
#         - fixed small bug in RULE parser to correctly parse rules with
#           identical source and destination fields
#         - faster initial scanning of DEVICE fields
# v0.7  - Lubomir Bulej <pallas at kadan.cz>
#         - lots of various cleanups and reorganizations; the parsing is now
#           some 40% faster, but the class ID must be in range 0x0002-0xffff
#           (again). Because of the number of internal changes and the above
#           class ID restriction, I bumped the version to 0.7 to indicate
#           something might have got broken :)
#         - changed PRIO_{U32,FW,ROUTE} to PRIO_{RULE,MARK,REALM}
#           for consistency with filter keywords
#         - exposed "compile" command
#       - Catalin Petrescu <taz at dntis.ro>
#         - support for port masks in RULE (u32) filter
#       - Jordan Vrtanoski <obeliks at mt.net.mk>
#         - support for week days in TIME rules
# v0.6.4- Lubomir Bulej <pallas at kadan.cz>
#         - added PRIO_* variables to allow easy control of filter priorities
#         - added caching to speed up CBQ start, the cache is invalidated
#           whenever any of the configuration files changes
#         - updated the readme section + some cosmetic fixes
```

```
# v0.6.3- Lubomir Bulej <pallas at kadan.cz>
#         - removed setup of (unnecessary) class 1:1 - all classes
#           now use qdisc's default class 1:0 as their parent
#         - minor fix in the timecheck branch - classes
#           without leaf qdisc were not updated
#         - minor fix to avoid timecheck failure when run
#           at time with minutes equal to 08 or 09
#         - respect CBQ_PATH setting in environment
#         - made PRIO=5 default, rendering it optional in configs
#         - added support for route filter, see notes about REALM keyword
#         - added support for fw filter, see notes about MARK keyword
#         - added filter display to "list" and "stats" commands
#         - readme section update + various cosmetic fixes
# v0.6.2- Catalin Petrescu <taz at dntis.ro>
#         - added tunnels interface handling
# v0.6.1- Pavel Golubev <pg at ksi-linux.com>
#         - added sch_prio module loading
#           (thanks johan at iglo.virtual.or.id for reminding)
#         - resolved errors resulting from stricter syntax checking in bash2
#       - Lubomir Bulej <pallas at kadan.cz>
#         - various cosmetic fixes
# v0.6  - Lubomir Bulej <pallas at kadan.cz>
#         - attempt to limit number of spawned processes by utilizing
#           more of sed power (use sed instead of grep+cut)
#         - simplified TIME parser, using bash builtins
#         - added initial support for SFQ as leaf qdisc
#         - reworked the documentation part a little
#         - incorporated pending patches and ideas submitted by
#           following people for versions 0.3 into version 0.6
#       - Miguel Freitas <miguel at cetuc.puc-rio.br>
#         - in case of overlapping TIME parameters, the last match is taken
#       - Juanjo Ciarlante <jjo at mendoza.gov.ar>
#         - chkconfig tags, list + stats startup parameters
#         - optional tc & ip command logging (into /var/run/cbq-*)
#       - Rafal Maszkowski <rzm at icm.edu.pl>
#         - PEAK parameter for setting TBF's burst peak rate
#         - fix for many config files (use find instead of ls)
# v0.5.1- Lubomir Bulej <pallas at kadan.cz>
#         - fixed little but serious bug in RULE parser
# v0.5  - Lubomir Bulej <pallas at kadan.cz>
#         - added options PARENT, LEAF, ISOLATED and BOUNDED. This allows
#           (with some attention to config file ordering) for creating
#           hierarchical structures of shapers with classes able (or unable)
#           to borrow bandwidth from their parents.
#         - class ID check allows hexadecimal numbers
#         - rewritten & simplified RULE parser
#         - cosmetic changes to improve readability
#         - reorganization to avoid duplicate code (timecheck etc.)
#         - timecheck doesn't check classes without TIME fields anymore
# v0.4  - Lubomir Bulej <pallas at kadan.cz>
#         - small bugfix in RULE parsing code
#         - simplified configuration parsing code
#         - several small cosmetic changes
#         - TIME parameter can be now specified more than once allowing you to
#           differentiate RATE throughout the whole day. Time overlapping is
#           not checked, first match is taken. Midnight wrap (eg. 20:00-6:00)
#           is allowed and taken care of.
# v0.3a4- fixed small bug in IF operator. Thanks to
#         Rafal Maszkowski <rzm at icm.edu.pl>
# v0.3a3- fixed grep bug when using more than 10 eth devices. Thanks to David
#         Trcka <trcka at poda.cz>.
# v0.3a2- fixed bug in "if" operator. Thanks kad at dgtu.donetsk.ua.
# v0.3a - added TIME parameter. Example: TIME=00:00-19:00;64Kbit/6Kbit
#         So, between 00:00 and 19:00 the RATE will be 64Kbit.
#         Just start "cbq.init timecheck" periodically from cron
#         (every 10 minutes for example). DON'T FORGET though, to run
#         "cbq.init start" for CBQ to initialize.
# v0.2  - Some cosmetic changes. Now it is more compatible with old bash
#         version. Thanks to Stanislav V. Voronyi <stas at cnti.uanet.kharkov.ua>.
# v0.1  - First public release
#
#
# README
# ------
#
# First of all - this is just a SIMPLE EXAMPLE of CBQ power.
# Don't ask me "why" and "how" :)
#
# This script is meant to simplify setup and management of relatively simple
# CBQ-based traffic control on Linux. Access to advanced networking features
# of Linux kernel is provided by "ip" and "tc" utilities from A. Kuznetsov's
# iproute2 package, available at ftp://ftp.inr.ac.ru/ip-routing. Because the
# utilities serve primarily to translate user wishes to RTNETLINK commands,
```

```
# their interface is rather spartan, intolerant and requires quite a lot of
# typing. And typing is what this script attempts to reduce :)
#
# The advanced networking stuff in Linux is pretty flexible and this script
# aims to bring some of its features to the not-so-hard-core Linux users. Of
# course, there is a tradeoff between simplicity and flexibility and you may
# realize that the flexibility suffered too much for your needs -- time to
# face "ip" and "tc" interface.
#
# To speed up the "start" command, simple caching was introduced in version
# 0.6.4. The caching works so that the sequence of "tc" commands for given
# configuration is stored in a file (/var/cache/cbq.init by default) which
# is used next time the "start" command is run to avoid repeated parsing of
# configuration files. This cache is invalidated whenever any of the CBQ
# configuration files changes. If you want to run "cbq.init start" without
# caching, run it as "cbq.init start nocache". If you want to force cache
# invalidation, run it as "cbq.init start invalidate". Caching is disabled
# if you have logging enabled (ie. CBQ_DEBUG is not empty).
#
# If you only want cqb.init to translate your configuration to "tc" commands,
# use "compile" command which will output "tc" commands required to build
# your configuration. Bear in mind that "compile" does not check if the "tc"
# commands were successful - this is done (in certain places) only when the
# "start nocache" command is used, which is also useful when creating the
# configuration to check whether it is completely valid.
#
# All CBQ parameters are valid for Ethernet interfaces only, The script was
# tested on various Linux kernel versions from series 2.1 to 2.4 and several
# distributions with KSI Linux (Nostromo version) as the premier one.
#
#
# HOW DOES IT WORK?
# -----------------
#
# Every traffic class must be described by a file in the $CBQ_PATH directory
# (/etc/sysconfig/cbq by default) - one file per class.
#
# The config file names must obey mandatory format: cbq-<clsid>.<name> where
# <clsid> is two-byte hexadecimal number in range <0002-FFFF> (which in fact
# is a CBQ class ID) and <name> is the name of the class -- anything to help
# you distinguish the configuration files. For small amount of classes it is
# often possible (and convenient) to let <clsid> resemble bandwidth of the
# class.
#
# Example of valid config name:
#       cbq-1280.My_first_shaper
#
#
# The configuration file may contain the following parameters:
#
### Device parameters
#
# DEVICE=<ifname>,<bandwidth>[,<weight>]     mandatory
# DEVICE=eth0,10Mbit,1Mbit
#
#       <ifname> is the name of the interface you want to control
#               traffic on, e.g. eth0
#       <bandwidth> is the physical bandwidth of the device, e.g. for
#               ethernet 10Mbit or 100Mbit, for arcnet 2Mbit
#       <weight> is tuning parameter that should be proportional to
#               <bandwidth>. As a rule of thumb: <weight> = <bandwidth> / 10
#
# When you have more classes on one interface, it is enough to specify
# <bandwidth> [and <weight>] only once, therefore in other files you only
# need to set DEVICE=<ifname>.
#
### Class parameters
#
# RATE=<speed>                               mandatory
# RATE=5Mbit
#
#       Bandwidth allocated to the class. Traffic going through the class is
#       shaped to conform to specified rate. You can use Kbit, Mbit or bps,
#       Kbps and Mbps as suffices. If you don't specify any unit, bits/sec
#       are used. Also note that "bps" means "bytes per second", not bits.
#
# WEIGHT=<speed>                             mandatory
# WEIGHT=500Kbit
#
#       Tuning parameter that should be proportional to RATE. As a rule
#       of thumb, use WEIGHT ~= RATE / 10.
#
# PRIO=<1-8>                                 optional, default 5
```

```
# PRIO=5
#
#       Priority of class traffic. The higher the number, the lesser
#       the priority. Priority of 5 is just fine.
#
# PARENT=<clsid>                                optional, default not set
# PARENT=1280
#
#       Specifies ID of the parent class to which you want this class be
#       attached. You might want to use LEAF=none for the parent class as
#       mentioned below. By using this parameter and carefully ordering the
#       configuration files, it is possible to create simple hierarchical
#       structures of CBQ classes. The ordering is important so that parent
#       classes are constructed prior to their children.
#
# LEAF=none|tbf|sfq                             optional, default "tbf"
#
#       Tells the script to attach specified leaf queueing discipline to CBQ
#       class. By default, TBF is used. Note that attaching TBF to CBQ class
#       shapes the traffic to conform to TBF parameters and prevents the class
#       from borrowing bandwidth from its parent even if you have BOUNDED set
#       to "no". To allow the class to borrow bandwith (provided it is not
#       bounded), you must set LEAF to "none" or "sfq".
#
#       If you want to ensure (approximately) fair sharing of bandwidth among
#       several hosts in the same class, you might want to specify LEAF=sfq to
#       attach SFQ as leaf queueing discipline to that class.
#
# BOUNDED=yes|no                                optional, default "yes"
#
#       If set to "yes", the class is not allowed to borrow bandwidth from
#       its parent class in overlimit situation. If set to "no", the class
#       will be allowed to borrow bandwidth from its parent.
#
# Note: Don't forget to set LEAF to "none" or "sfq", otherwise the class will
#       have TBF attached to itself and will not be able to borrow unused
#       bandwith from its parent.
#
# ISOLATED=yes|no                               optional, default "no"
#
#       If set to "yes", the class will not lend unused bandwidth to
#       its children.
#
### TBF qdisc parameters
#
# BUFFER=<bytes>[/<bytes>]                      optional, default "10Kb/8"
#
#       This parameter controls the depth of the token bucket. In other
#       words it represents the maximal burst size the class can send.
#       The optional part of parameter is used to determine the length
#       of intervals in packet sizes, for which the transmission times
#       are kept.
#
# LIMIT=<bytes>                                 optional, default "15Kb"
#
#       This parameter determines the maximal length of backlog. If
#       the queue contains more data than specified by LIMIT, the
#       newly arriving packets are dropped. The length of backlog
#       determines queue latency in case of congestion.
#
# PEAK=<speed>                                  optional, default not set
#
#       Maximal peak rate for short-term burst traffic. This allows you
#       to control the absolute peak rate the class can send at, because
#       single TBF that allows 256Kbit/s would of course allow rate of
#       512Kbit for half a second or 1Mbit for a quarter of second.
#
# MTU=<bytes>                                   optional, default "1500"
#
#       Maximum number of bytes that can be sent at once over the
#       physical medium. This parameter is required when you specify
#       PEAK parameter. It defaults to MTU of ethernet - for other
#       media types you might want to change it.
#
# Note: Setting TBF as leaf qdisc will effectively prevent the class from
#       borrowing bandwidth from the ancestor class, because even if the
#       class allows more traffic to pass through, it is then shaped to
#       conform to TBF.
#
### SFQ qdisc parameters
#
# The SFQ queueing discipline is a cheap way for sharing class bandwidth
# among several hosts. As it is stochastic, the fairness is approximate but
```

```
# it will do the job in most cases. If you want real fairness, you should
# probably use WRR (weighted round robin) or WFQ queueing disciplines. Note
# that SFQ does not do any traffic shaping - the shaping is done by the CBQ
# class the SFQ is attached to.
#
# QUANTUM=<bytes>                               optional, default not set
#
#       This parameter should not be set lower than link MTU, for ethernet
#       it is 1500b, or (with MAC header) 1514b which is the value used
#       in Alexey Kuznetsov's examples.
#
# PERTURB=<seconds>                             optional, default "10"
#
#       Period of hash function perturbation. If unset, hash reconfiguration
#       will never take place which is what you probably don't want. The
#       default value of 10 seconds is probably a good one.
#
### Filter parameters
#
# RULE=[[saddr[/prefix]][:port[/mask]],][daddr[/prefix]][:port[/mask]]
#
#       These parameters make up "u32" filter rules that select traffic for
#       each of the classes. You can use multiple RULE fields per config.
#
#       The optional port mask should only be used by advanced users who
#       understand how the u32 filter works.
#
# Some examples:
#
#       RULE=10.1.1.0/24:80
#              selects traffic going to port 80 in network 10.1.1.0
#
#       RULE=10.2.2.5
#              selects traffic going to any port on single host 10.2.2.5
#
#       RULE=10.2.2.5:20/0xfffe
#              selects traffic going to ports 20 and 21 on host 10.2.2.5
#
#       RULE=:25,10.2.2.128/26:5000
#              selects traffic going from anywhere on port 50 to
#              port 5000 in network 10.2.2.128
#
#       RULE=10.5.5.5:80,
#              selects traffic going from port 80 of single host 10.5.5.5
#
#
#
# REALM=[srealm,][drealm]
#
#       These parameters make up "route" filter rules that classify traffic
#       according to packet source/destination realms. For information about
#       realms, see Alexey Kuznetsov's IP Command Reference. This script
#       does not define any realms, it justs builds "tc filter" commands
#       for you if you need to classify traffic this way.
#
#       Realm is either a decimal number or a string referencing entry in
#       /etc/iproute2/rt_realms (usually).
#
# Some examples:
#
#       REALM=russia,internet
#              selects traffic going from realm "russia" to realm "internet"
#
#       REALM=freenet,
#              selects traffic going from realm "freenet"
#
#       REALM=10
#              selects traffic going to realm 10
#
#
#
# MARK=<mark>
#
#       These parameters make up "fw" filter rules that select traffic for
#       each of the classes accoring to firewall "mark". Mark is a decimal
#       number packets are tagged with if firewall rules say so. You can
#       use multiple MARK fields per config.
#
#
# Note: Rules for different filter types can be combined. Attention must be
#       paid to the priority of filter rules, which can be set below using
#       PRIO_{RULE,MARK,REALM} variables.
#
```

```
### Time ranging parameters
#
# TIME=[<dow>,<dow>, ...,<dow>/]<from>-<till>;<rate>/<weight>[/<peak>]
# TIME=0,1,2,5/18:00-06:00;256Kbit/25Kbit
# TIME=60123/18:00-06:00;256Kbit/25Kbit
# TIME=18:00-06:00;256Kbit/25Kbit
#
#       This parameter allows you to differentiate the class bandwidth
#       throughout the day. You can specify multiple TIME parameters, if
#       the times overlap, last match is taken. The fields <rate>, <weight>
#       and <peak> correspond to parameters RATE, WEIGHT and PEAK (which
#       is optional and applies to TBF leaf qdisc only).
#
#       You can also specify days of week when the TIME rule applies. <dow>
#       is numeric, 0 corresponds to sunday, 1 corresponds to monday, etc.
#
###
#
# Sample configuration file: cbq-1280.My_first_shaper
#
# ------------------------------------------------------------------------
# DEVICE=eth0,10Mbit,1Mbit
# RATE=128Kbit
# WEIGHT=10Kbit
# PRIO=5
# RULE=192.128.1.0/24
# ------------------------------------------------------------------------
#
# The configuration says that we will control traffic on 10Mbit ethernet
# device eth0 and the traffic going to network 192.168.1.0 will be
# processed with priority 5 and shaped to rate of 128Kbit.
#
# Note that you can control outgoing traffic only. If you want to control
# traffic in both directions, you must set up CBQ for both interfaces.
#
# Consider the following example:
#
#                   +---------+     192.168.1.1
# BACKBONE -----eth0-|  linux  |-eth1------*-[client]
#                   +---------+
#
# Imagine you want to shape traffic from backbone to the client to 28Kbit
# and traffic in the opposite direction to 128Kbit. You need to setup CBQ
# on both eth0 and eth1 interfaces, thus you need two config files:
#
# cbq-028.backbone-client
# ------------------------------------------------------------------------
# DEVICE=eth1,10Mbit,1Mbit
# RATE=28Kbit
# WEIGHT=2Kbit
# PRIO=5
# RULE=192.168.1.1
# ------------------------------------------------------------------------
#
# cbq-128.client-backbone
# ------------------------------------------------------------------------
# DEVICE=eth0,10Mbit,1Mbit
# RATE=128Kbit
# WEIGHT=10Kbit
# PRIO=5
# RULE=192.168.1.1,
# ------------------------------------------------------------------------
#
# Pay attention to comma "," in the RULE field - it denotes source address!
#
# Enjoy.
#
################################################################################

export LC_ALL=C

### Command locations
TC=/sbin/tc
IP=/sbin/ip
MP=/sbin/modprobe

### Default filter priorities (must be different)
PRIO_RULE_DEFAULT=${PRIO_RULE:-100}
PRIO_MARK_DEFAULT=${PRIO_MARK:-200}
PRIO_REALM_DEFAULT=${PRIO_REALM:-300}

### Default CBQ_PATH & CBQ_CACHE settings
CBQ_PATH=${CBQ_PATH:-/etc/cbq}
```

```
CBQ_CACHE=${CBQ_CACHE:-/var/cache/cbq.init}

### Uncomment to enable logfile for debugging
#CBQ_DEBUG="/var/run/cbq-$1"

### Modules to probe for. Uncomment the last CBQ_PROBE
### line if you have QoS support compiled into kernel
CBQ_PROBE="sch_cbq sch_tbf sch_sfq sch_prio"
CBQ_PROBE="$CBQ_PROBE cls_fw cls_u32 cls_route"
#CBQ_PROBE=""

### Keywords required for qdisc & class configuration
CBQ_WORDS="DEVICE|RATE|WEIGHT|PRIO|PARENT|LEAF|BOUNDED|ISOLATED"
CBQ_WORDS="$CBQ_WORDS|PRIO_MARK|PRIO_RULE|PRIO_REALM|BUFFER"
CBQ_WORDS="$CBQ_WORDS|LIMIT|PEAK|MTU|QUANTUM|PERTURB"


################################################################################
############################## SUPPORT FUNCTIONS ###############################
################################################################################

### Get list of network devices
cbq_device_list () {
        ip link show| sed -n "/^[0-9]/ \
                { s/^[0-9]\+: \([a-z0-9._]\+\)[:@].*/\1/; p; }"
} # cbq_device_list


### Remove root class from device $1
cbq_device_off () {
        tc qdisc del dev $1 root 2> /dev/null
} # cbq_device_off


### Remove CBQ from all devices
cbq_off () {
        for dev in `cbq_device_list`; do
                cbq_device_off $dev
        done
} # cbq_off


### Prefixed message
cbq_message () {
        echo -e "**CBQ: $@"
} # cbq_message

### Failure message
cbq_failure () {
        cbq_message "$@"
        exit 1
} # cbq_failure

### Failure w/ cbq-off
cbq_fail_off () {
        cbq_message "$@"
        cbq_off
        exit 1
} # cbq_fail_off


### Convert time to absolute value
cbq_time2abs () {
        local min=${1##*:}; min=${min##0}
        local hrs=${1%%:*}; hrs=${hrs##0}
        echo $[hrs*60 + min]
} # cbq_time2abs


### Display CBQ setup
cbq_show () {
        for dev in `cbq_device_list`; do
                [ `tc qdisc show dev $dev| wc -l` -eq 0 ] && continue
                echo -e "### $dev: queueing disciplines\n"
                tc $1 qdisc show dev $dev; echo

                [ `tc class show dev $dev| wc -l` -eq 0 ] && continue
                echo -e "### $dev: traffic classes\n"
                tc $1 class show dev $dev; echo

                [ `tc filter show dev $dev| wc -l` -eq 0 ] && continue
                echo -e "### $dev: filtering rules\n"
                tc $1 filter show dev $dev; echo
```

```
        done
} # cbq_show


### Check configuration and load DEVICES, DEVFIELDS and CLASSLIST from $1
cbq_init () {
        ### Get a list of configured classes
        CLASSLIST=`find $1 \( -type f -or -type l \) -name 'cbq-*' \
                -not -name '*~' -maxdepth 1 -printf "%f\n"| sort`
        [ -z "$CLASSLIST" ] &&
                cbq_failure "no configuration files found in $1!"

        ### Gather all DEVICE fields from $1/cbq-*
        DEVFIELDS=`find $1 \( -type f -or -type l \) -name 'cbq-*' \
                -not -name '*~' -maxdepth 1| xargs sed -n 's/#.*//; \
                s/[[:space:]]//g; /^DEVICE=[^,]*,[^,]*\(,[^,]*\)\?/ \
                { s/.*=//; p; }'| sort -u`
        [ -z "$DEVFIELDS" ] &&
                cbq_failure "no DEVICE field found in $1/cbq-*!"

        ### Check for different DEVICE fields for the same device
        DEVICES=`echo "$DEVFIELDS"| sed 's/,.*//'| sort -u`
        [ `echo "$DEVICES"| wc -l` -ne `echo "$DEVFIELDS"| wc -l` ] &&
                cbq_failure "different DEVICE fields for single device!\n$DEVFIELDS"
} # cbq_init


### Load class configuration from $1/$2
cbq_load_class () {
        CLASS=`echo $2| sed 's/^cbq-0*//; s/^\([0-9a-fA-F]\+\).*/\1/'`
        CFILE=`sed -n 's/#.*//; s/[[:space:]]//g; /^[[:alnum:]_]\+=[[:alnum:].,:;/*@-_]\
+$/ p' $1/$2`

        ### Check class number
        IDVAL=`/usr/bin/printf "%d" 0x$CLASS 2> /dev/null`
        [ $? -ne 0 -o $IDVAL -lt 2 -o $IDVAL -gt 65535 ] &&
                cbq_fail_off "class ID of $2 must be in range <0002-FFFF>!"

        ### Set defaults & load class
        RATE=""; WEIGHT=""; PARENT=""; PRIO=5
        LEAF=tbf; BOUNDED=yes; ISOLATED=no
        BUFFER=10Kb/8; LIMIT=15Kb; MTU=1500
        PEAK=""; PERTURB=10; QUANTUM=""

        PRIO_RULE=$PRIO_RULE_DEFAULT
        PRIO_MARK=$PRIO_MARK_DEFAULT
        PRIO_REALM=$PRIO_REALM_DEFAULT

        eval `echo "$CFILE"| grep -E "^($CBQ_WORDS)="`

        ### Require RATE/WEIGHT
        [ -z "$RATE" -o -z "$WEIGHT" ] &&
                cbq_fail_off "missing RATE or WEIGHT in $2!"

        ### Class device
        DEVICE=${DEVICE%%,*}
        [ -z "$DEVICE" ] && cbq_fail_off "missing DEVICE field in $2!"

        BANDWIDTH=`echo "$DEVFIELDS"| sed -n "/^$DEVICE,/ \
                { s/[^,]*,\([^,]*\).*/\1/; p; q; }"`

        ### Convert to "tc" options
        PEAK=${PEAK:+peakrate $PEAK}
        PERTURB=${PERTURB:+perturb $PERTURB}
        QUANTUM=${QUANTUM:+quantum $QUANTUM}

        [ "$BOUNDED" = "no" ] && BOUNDED="" || BOUNDED="bounded"
        [ "$ISOLATED" = "yes" ] && ISOLATED="isolated" || ISOLATED=""
} # cbq_load_class


################################################################################
##################################### INIT #####################################
################################################################################

### Check for presence of ip-route2 in usual place
[ -x $TC -a -x $IP ] ||
        cbq_failure "ip-route2 utilities not installed or executable!"


### ip/tc wrappers
if [ "$1" = "compile" ]; then
        ### no module probing
```

```
        CBQ_PROBE=""

        ip () {
                $IP "$@"
        } # ip

        ### echo-only version of "tc" command
        tc () {
                echo "$TC $@"
        } # tc

elif [ -n "$CBQ_DEBUG" ]; then
        echo -e "# `date`" > $CBQ_DEBUG

        ### Logging version of "ip" command
        ip () {
                echo -e "\n# ip $@" >> $CBQ_DEBUG
                $IP "$@" 2>&1 | tee -a $CBQ_DEBUG
        } # ip

        ### Logging version of "tc" command
        tc () {
                echo -e "\n# tc $@" >> $CBQ_DEBUG
                $TC "$@" 2>&1 | tee -a $CBQ_DEBUG
        } # tc
else
        ### Default wrappers

        ip () {
                $IP "$@"
        } # ip

        tc () {
                $TC "$@"
        } # tc
fi # ip/tc wrappers


case "$1" in

################################################################################
################################ START/COMPILE #################################
################################################################################

start|compile)

### Probe QoS modules (start only)
for module in $CBQ_PROBE; do
        $MP $module || cbq_failure "failed to load module $module"
done

### If we are in compile/nocache/logging mode, don't bother with cache
if [ "$1" != "compile" -a "$2" != "nocache" -a -z "$CBQ_DEBUG" ]; then
        VALID=1

        ### validate the cache
        [ "$2" = "invalidate" -o ! -f $CBQ_CACHE ] && VALID=0
        if [ $VALID -eq 1 ]; then
                [ `find $CBQ_PATH -maxdepth 1 -newer $CBQ_CACHE| \
                  wc -l` -gt 0 ] && VALID=0
        fi

        ### compile the config if the cache is invalid
        if [ $VALID -ne 1 ]; then
                $0 compile > $CBQ_CACHE ||
                        cbq_fail_off "failed to compile CBQ configuration!"
        fi

        ### run the cached commands
        exec /bin/sh $CBQ_CACHE 2> /dev/null
fi

### Load DEVICES, DEVFIELDS and CLASSLIST
cbq_init $CBQ_PATH


### Setup root qdisc on all configured devices
for dev in $DEVICES; do
        ### Retrieve device bandwidth and, optionally, weight
        DEVTEMP=`echo "$DEVFIELDS"| sed -n "/^$dev,/ { s/$dev,//; p; q; }"`
        DEVBWDT=${DEVTEMP%%,*};       DEVWGHT=${DEVTEMP##*,}
        [ "$DEVBWDT" = "$DEVWGHT" ] && DEVWGHT=""
```

```
        ### Device bandwidth is required
        if [ -z "$DEVBWDT" ]; then
                cbq_message "could not determine bandwidth for device $dev!"
                cbq_failure "please set up the DEVICE fields properly!"
        fi

        ### Check if the device is there
        ip link show $dev &> /dev/null ||
                cbq_fail_off "device $dev not found!"

        ### Remove old root qdisc from device
        cbq_device_off $dev


        ### Setup root qdisc + class for device
        tc qdisc add dev $dev root handle 1 cbq \
        bandwidth $DEVBWDT avpkt 1000 cell 8

        ### Set weight of the root class if set
        [ -n "$DEVWGHT" ] &&
                tc class change dev $dev root cbq weight $DEVWGHT allot 1514

        [ "$1" = "compile" ] && echo
done # dev


### Setup traffic classes
for classfile in $CLASSLIST; do
        cbq_load_class $CBQ_PATH $classfile

        ### Create the class
        tc class add dev $DEVICE parent 1:$PARENT classid 1:$CLASS cbq \
        bandwidth $BANDWIDTH rate $RATE weight $WEIGHT prio $PRIO \
        allot 1514 cell 8 maxburst 20 avpkt 1000 $BOUNDED $ISOLATED ||
                cbq_fail_off "failed to add class $CLASS with parent $PARENT on $DEVICE!"

        ### Create leaf qdisc if set
        if [ "$LEAF" = "tbf" ]; then
                tc qdisc add dev $DEVICE parent 1:$CLASS handle $CLASS tbf \
                rate $RATE buffer $BUFFER limit $LIMIT mtu $MTU $PEAK
        elif [ "$LEAF" = "sfq" ]; then
                tc qdisc add dev $DEVICE parent 1:$CLASS handle $CLASS sfq \
                $PERTURB $QUANTUM
        fi


        ### Create fw filter for MARK fields
        for mark in `echo "$CFILE"| sed -n '/^MARK/ { s/.*=//; p; }'`; do
                ### Attach fw filter to root class
                tc filter add dev $DEVICE parent 1:0 protocol ip \
                prio $PRIO_MARK handle $mark fw classid 1:$CLASS
        done ### mark

        ### Create route filter for REALM fields
        for realm in `echo "$CFILE"| sed -n '/^REALM/ { s/.*=//; p; }'`; do
                ### Split realm into source & destination realms
                SREALM=${realm%%,*}; DREALM=${realm##*,}
                [ "$SREALM" = "$DREALM" ] && SREALM=""

                ### Convert asterisks to empty strings
                SREALM=${SREALM#\*}; DREALM=${DREALM#\*}

                ### Attach route filter to the root class
                tc filter add dev $DEVICE parent 1:0 protocol ip \
                prio $PRIO_REALM route ${SREALM:+from $SREALM} \
                ${DREALM:+to $DREALM} classid 1:$CLASS
        done ### realm

        ### Create u32 filter for RULE fields
        for rule in `echo "$CFILE"| sed -n '/^RULE/ { s/.*=//; p; }'`; do
                ### Split rule into source & destination
                SRC=${rule%%,*}; DST=${rule##*,}
                [ "$SRC" = "$rule" ] && SRC=""


                ### Split destination into address, port & mask fields
                DADDR=${DST%%:*}; DTEMP=${DST##*:}
                [ "$DADDR" = "$DST" ] && DTEMP=""

                DPORT=${DTEMP%%/*}; DMASK=${DTEMP##*/}
                [ "$DPORT" = "$DTEMP" ] && DMASK="0xffff"
```

```
                ### Split up source (if specified)
                SADDR=""; SPORT=""
                if [ -n "$SRC" ]; then
                        SADDR=${SRC%%:*}; STEMP=${SRC##*:}
                        [ "$SADDR" = "$SRC" ] && STEMP=""

                        SPORT=${STEMP%%/*}; SMASK=${STEMP##*/}
                        [ "$SPORT" = "$STEMP" ] && SMASK="0xffff"
                fi


                ### Convert asterisks to empty strings
                SADDR=${SADDR#\*}; DADDR=${DADDR#\*}

                ### Compose u32 filter rules
                u32_s="${SPORT:+match ip sport $SPORT $SMASK}"
                u32_s="${SADDR:+match ip src $SADDR} $u32_s"
                u32_d="${DPORT:+match ip dport $DPORT $DMASK}"
                u32_d="${DADDR:+match ip dst $DADDR} $u32_d"

                ### Uncomment the following if you want to see parsed rules
                #echo "$rule: $u32_s $u32_d"

                ### Attach u32 filter to the appropriate class
                tc filter add dev $DEVICE parent 1:0 protocol ip \
                prio $PRIO_RULE u32 $u32_s $u32_d classid 1:$CLASS
        done ### rule

        [ "$1" = "compile" ] && echo
done ### classfile
;;


#############################################################################
################################# TIME CHECK ################################
#############################################################################

timecheck)

### Get time + weekday
TIME_TMP=`date +%w/%k:%M`
TIME_DOW=${TIME_TMP%%/*}
TIME_NOW=${TIME_TMP##*/}

### Load DEVICES, DEVFIELDS and CLASSLIST
cbq_init $CBQ_PATH

### Run through all classes
for classfile in $CLASSLIST; do
        ### Gather all TIME rules from class config
        TIMESET=`sed -n 's/#.*//; s/[[:space:]]//g; /^TIME/ { s/.*=//; p; }' \
                $CBQ_PATH/$classfile`
        [ -z "$TIMESET" ] && continue

        MATCH=0; CHANGE=0
        for timerule in $TIMESET; do
                TIME_ABS=`cbq_time2abs $TIME_NOW`

                ### Split TIME rule to pieces
                TIMESPEC=${timerule%%;*}; PARAMS=${timerule##*;}
                WEEKDAYS=${TIMESPEC%%/*}; INTERVAL=${TIMESPEC##*/}
                BEG_TIME=${INTERVAL%%-*}; END_TIME=${INTERVAL##*-}

                ### Check the day-of-week (if present)
                [ "$WEEKDAYS" != "$INTERVAL" -a \
                  -n "${WEEKDAYS##*$TIME_DOW*}" ] && continue

                ### Compute interval boundaries
                BEG_ABS=`cbq_time2abs $BEG_TIME`
                END_ABS=`cbq_time2abs $END_TIME`

                ### Midnight wrap fixup
                if [ $BEG_ABS -gt $END_ABS ]; then
                        [ $TIME_ABS -le $END_ABS ] &&
                                TIME_ABS=$[TIME_ABS + 24*60]

                        END_ABS=$[END_ABS + 24*60]
                fi

                ### If the time matches, remember params and set MATCH flag
                if [ $TIME_ABS -ge $BEG_ABS -a $TIME_ABS -lt $END_ABS ]; then
                        TMP_RATE=${PARAMS%%/*}; PARAMS=${PARAMS#*/}
                        TMP_WGHT=${PARAMS%%/*}; TMP_PEAK=${PARAMS##*/}
```

```
                        [ "$TMP_PEAK" = "$TMP_WGHT" ] && TMP_PEAK=""
                        TMP_PEAK=${TMP_PEAK:+peakrate $TMP_PEAK}

                        MATCH=1
                fi
        done ### timerule


        cbq_load_class $CBQ_PATH $classfile

        ### Get current RATE of CBQ class
        RATE_NOW=`tc class show dev $DEVICE| sed -n \
                "/cbq 1:$CLASS / { s/.*rate //; s/ .*//; p; q; }"`
        [ -z "$RATE_NOW" ] && continue

        ### Time interval matched
        if [ $MATCH -ne 0 ]; then

                ### Check if there is any change in class RATE
                if [ "$RATE_NOW" != "$TMP_RATE" ]; then
                        NEW_RATE="$TMP_RATE"
                        NEW_WGHT="$TMP_WGHT"
                        NEW_PEAK="$TMP_PEAK"
                        CHANGE=1
                fi

        ### Match not found, reset to default RATE if necessary
        elif [ "$RATE_NOW" != "$RATE" ]; then
                NEW_WGHT="$WEIGHT"
                NEW_RATE="$RATE"
                NEW_PEAK="$PEAK"
                CHANGE=1
        fi

        ### If there are no changes, go for next class
        [ $CHANGE -eq 0 ] && continue

        ### Replace CBQ class
        tc class replace dev $DEVICE classid 1:$CLASS cbq \
        bandwidth $BANDWIDTH rate $NEW_RATE weight $NEW_WGHT prio $PRIO \
        allot 1514 cell 8 maxburst 20 avpkt 1000 $BOUNDED $ISOLATED

        ### Replace leaf qdisc (if any)
        if [ "$LEAF" = "tbf" ]; then
                tc qdisc replace dev $DEVICE handle $CLASS tbf \
                rate $NEW_RATE buffer $BUFFER limit $LIMIT mtu $MTU $NEW_PEAK
        fi

        cbq_message "$TIME_NOW: class $CLASS on $DEVICE changed rate ($RATE_NOW ->
$NEW_RATE)"
done ### class file
;;


###############################################################################
################################## THE REST ###################################
###############################################################################

stop)
        cbq_off
        ;;

list)
        cbq_show
        ;;

stats)
        cbq_show -s
        ;;

restart)
        shift
        $0 stop
        $0 start "$@"
        ;;

*)
        echo "Usage: `basename $0` {start|compile|stop|restart|timecheck|list|stats}"
esac
```

LAMPIRAN B

SKRIP KONFIGURASI

1. **Skenario dengan *leaf class* berdasarkan IP *address***
   a. **Rule untuk HTB**

```
RATE=512Kbit
LEAF=none
RULE=192.168.1.0/24
```

Rule 1 *File* /etc/sysconfig/htb/eth1-2.root

```
RATE=128Kbit
CEIL=512Kbit
LEAF=none
RULE=192.168.1.2
```

Rule 2 *File* /etc/sysconfig/htb/eth1-2:10.client1

```
RATE=128Kbit
CEIL=512Kbit
LEAF=none
RULE=192.168.1.3
```

Rule 3 *File* /etc/sysconfig/htb/eth1-2:20.client2

```
RATE=128Kbit
CEIL=512Kbit
LEAF=none
RULE=192.168.1.4
```

Rule 4 *File* /etc/sysconfig/htb/eth1-2:30.client3

```
RATE=128Kbit
CEIL=512Kbit
LEAF=none
RULE=192.168.1.5
```

Rule 5 *File* /etc/sysconfig/htb/eth1-2:40.client4

b. Rule untuk CBQ

```
DEVICE=eth1,100Mbit,10Mbit
RATE=512Kbit
WEIGHT=51.2Kbit
LEAF=none
RULE=192.168.1.0/24
```

Rule 6 *File* /etc/sysconfig/cbq/cbq-02.root

```
DEVICE=eth1
RATE=128Kbit
WEIGHT=12.8Kbit
PARENT=02
LEAF=none
BOUNDED=no
RULE=192.128.1.2
```

Rule 7 *File* /etc/sysconfig/cbq/cbq-03.client1

```
DEVICE=eth1
RATE=128Kbit
WEIGHT=12.8Kbit
PARENT=02
LEAF=none
BOUNDED=no
RULE=192.128.1.3
```

Rule 8 *File* /etc/sysconfig/cbq/cbq-04.client2

```
DEVICE=eth1
RATE=128Kbit
WEIGHT=12.8Kbit
PARENT=02
LEAF=none
BOUNDED=no
RULE=192.128.1.4
```

Rule 9 *File* /etc/sysconfig/cbq/cbq-05.client3

```
DEVICE=eth1
RATE=128Kbit
WEIGHT=12.8Kbit
PARENT=02
LEAF=none
BOUNDED=no
RULE=192.128.1.5
```

Rule 10 *File* /etc/sysconfig/cbq/cbq-06.client4

2. **Skenario dengan *leaf class* berdasarkan *port***
   a. **Rule skrip untuk HTB**

```
RATE=512Kbit
LEAF=none
RULE=192.168.1.0/24
```

Rule 11 *File* /etc/sysconfig/htb/eth1-2.root

```
RATE=128Kbit
CEIL=512Kbit
LEAF=none
RULE=192.168.1.0/24:3001
```

Rule 12 *File* /etc/sysconfig/htb/eth1-2:10.port1

```
RATE=128Kbit
CEIL=512Kbit
LEAF=none
RULE=192.168.1.0/24:3002
```

Rule 13 *File* /etc/sysconfig/htb/eth1-2:20. port2

```
RATE=128Kbit
CEIL=512Kbit
LEAF=none
RULE=192.168.1.0/24:3003
```

Rule 14 *File* /etc/sysconfig/htb/eth1-2:30.port3

```
RATE=128Kbit
CEIL=512Kbit
LEAF=none
RULE=192.168.1.0/24:3004
```

Rule 15 *File* /etc/sysconfig/htb/eth1-2:40.port4

   b. **Rule skrip untuk CBQ**

```
DEVICE=eth1,100Mbit,10Mbit
RATE=512Kbit
WEIGHT=51.2Kbit
LEAF=none
RULE=192.168.1.2
```

Rule 16 *File* /etc/sysconfig/cbq/cbq-02.root

```
DEVICE=eth0
RATE=128Kbit
WEIGHT=12.8Kbit
PARENT=02
LEAF=none
BOUNDED=no
RULE=192.128.1.0/24:3001
```

Rule 17 *File* /etc/sysconfig/cbq/cbq-03.port1

```
DEVICE=eth0
RATE=128Kbit
WEIGHT=12.8Kbit
PARENT=02
LEAF=none
BOUNDED=no
RULE=192.128.1.0/24:3002
```

Rule 18 *File* /etc/sysconfig/cbq/cbq-04.port2

```
DEVICE=eth0
RATE=128Kbit
WEIGHT=12.8Kbit
PARENT=.2
LEAF=none
BOUNDED=no
RULE=192.128.1.0/24:3003
```

Rule 19 *File* /etc/sysconfig/cbq/cbq-05.port3

```
DEVICE=eth0
RATE=128Kbit
WEIGHT=12.8Kbit
PARENT=02
LEAF=none
BOUNDED=no
RULE=192.128.1.0/24:3004
```

Rule 20 *File* /etc/sysconfig/cbq/cbq-06.port4

3. Skenario dengan parameter khusus

a. Rule skrip untuk HTB

```
RATE=512Kbit
LEAF=none
RULE=192.168.1.0/24
```

Rule 21 *File* /etc/sysconfig/htb/eth1-2.root

```
RATE=128Kbit
CEIL=128Kbit
LEAF=none
RULE=192.168.1.2
```

Rule 22 *File* /etc/sysconfig/htb/eth1-2:10.client1

```
RATE=128Kbit
CEIL=128Kbit
LEAF=none
RULE=192.168.1.3
```

Rule 23 *File* /etc/sysconfig/htb/eth1-2:20.client2

b. Rule skrip untuk CBQ

```
DEVICE=eth1,100Mbit,10Mbit
RATE=512Kbit
WEIGHT=51.2Kbit
LEAF=none
RULE=192.168.1.0/24
```

Rule 24 *File* /etc/sysconfig/cbq/cbq-02.root

```
DEVICE=eth0
RATE=128Kbit
WEIGHT=12.8Kbit
PARENT=02
LEAF=none
BOUNDED=yes
RULE=192.128.1.2
```

Rule 25 *File* /etc/sysconfig/cbq/cbq-03.client1

```
DEVICE=eth0
RATE=128Kbit
WEIGHT=12.8Kbit
PARENT=02
LEAF=none
BOUNDED=yes
RULE=192.128.1.3
```

Rule 26 *File* /etc/sysconfig/cbq/cbq-04.client2

2 LAMPIRAN C

3 HASIL PENGUJIAN

1. **Skenario dengan *leaf class* berdasarkan IP *address***
   a. **Skenario dengan satu *leaf class* berdasarkan IP *address***

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	17,2	499	23,249	0/732

2	1	17,3	498	24,075	0/732
3	1	17,3	498	24,080	0/732
Rata-Rata	1	17,267	498,333	23,801	0/732

Tabel 1 Parameter yang terukur untuk HTB IP 192.168.1.2

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	16,7	516	12,703	0/732
2	1	16,6	517	12,284	0/732
3	1	16,7	516	12,665	0/732
Rata-Rata	1	16,667	516,333	12,551	0/732

Tabel 2 Parameter yang terukur untuk CBQ IP 192.168.1.2

b. Skenario dengan dua *leaf class* berdasarkan IP *address*

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	33,6	256	48,196	0/732
2	1	33,6	256	41,712	0/732
3	1	33,8	255	46,459	0/732
Rata-Rata	1	33,667	255,667	45,456	0/732

Tabel 3 Parameter yang terukur untuk HTB IP 192.168.1.2

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	34,0	254	25,749	0/732
2	1	33,9	254	26,495	0/732
3	1	34,2	252	29,243	0/732
Rata-Rata	1	34,033	253,333	27,162	0/732

Tabel 4 Parameter yang terukur untuk HTB IP 192.168.1.3

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	32,0	262	78,643	8/732
2	1	32,0	260	62,583	24/732
3	1	32,0	261	57,686	13/732
Rata-Rata	1	32,000	261,000	66,304	15/732

Tabel 5 Parameter yang terukur untuk CBQ IP 192.168.1.2

Sampel ke -	Besar Pesan	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram

	(Mbyte)				
1	1	30,8	260	6,509	49/732
2	1	32,1	258	10,911	49/732
3	1	31,0	258	9,057	52/732
Rata-Rata	1	31,300	258,667	8,826	50/732

Tabel 6 Parameter yang terukur untuk CBQ IP 192.168.1.3

c. Skenario dengan tiga *leaf class* berdasarkan IP *address*

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	50,1	172	73,890	0/732
2	1	50,1	172	69,903	0/732
3	1	50,3	171	73,036	0/732
Rata-Rata	1	50,167	171,667	72,276	0/732

Tabel 7 Parameter yang terukur untuk HTB IP 192.168.1.2

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	51,0	169	54,786	0/732
2	1	51,0	169	51,994	0/732
3	1	51,1	168	53,238	0/732
Rata-Rata	1	51,033	168,667	53,339	0/732

Tabel 8 Parameter yang terukur untuk HTB IP 192.168.1.3

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	51,0	169	45,546	0/732
2	1	51,2	168	43,509	0/732
3	1	51,1	168	45,188	0/732
Rata-Rata	1	51,100	168,333	44,748	0/732

Tabel 9 Parameter yang terukur untuk HTB IP 192.168.1.4

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	32,3	192	56,991	202/732
2	1	32,2	187	59,686	220/732
3	1	32,6	185	89,801	218/732
Rata-Rata	1	32,367	188,000	68,826	213/732

Tabel 10 Parameter yang terukur untuk CBQ IP 192.168.1.2

Sampel ke -	Besar Pesan	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram

	(Mbyte)				
1	1	31,6	176	42,940	261/732
2	1	31,5	175	39,286	261/732
3	1	31,4	174	34,543	267/732
Rata-Rata	1	31,500	175,000	38,923	263/732

Tabel 11 Parameter yang terukur untuk CBQ IP 192.168.1.3

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	29,0	174	5,559	303/732
2	1	29,1	178	5,735	291/732
3	1	29,3	176	12,300	294/732
Rata-Rata	1	29,133	176,000	7,865	296/732

Tabel 12 Parameter yang terukur untuk CBQ IP 192.168.1.4

d. Skenario dengan satu *leaf class* berdasarkan IP *address*

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	65,0	132	73,800	0/732
2	1	62,3	138	72,761	0/732
3	1	62,1	139	80,920	0/732
Rata-Rata	1	63,133	136,333	75,827	0/732

Tabel 13 Parameter yang terukur untuk HTB IP 192.168.1.2

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	65,9	131	61,383	0/732
2	1	63,3	136	59,709	0/732
3	1	63,0	137	54,498	0/732
Rata-Rata	1	64,067	134,667	58,530	0/732

Tabel 14 Parameter yang terukur untuk HTB IP 192.168.1.3

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	65,9	131	51,948	0/732
2	1	63,2	136	52,563	0/732
3	1	63,0	137	52,161	0/732
Rata-Rata	1	64,033	134,667	52,224	0/732

Tabel 15 Parameter yang terukur untuk HTB IP 192.168.1.4

Sampel ke -	Besar Pesan	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram

	(Mbyte)				
1	1	57,3	125	97,041	125/732
2	1	46,9	125	97,339	0/732
3	1	45,4	125	97,170	0/732
Rata-Rata	1	49,867	125,000	97,183	42/732

Tabel 16 Parameter yang terukur untuk HTB IP 192.168.1.5

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	33,2	147	112,893	308/732
2	1	33,3	147	138,696	314/732
3	1	32,8	153	91,702	294/732
Rata-Rata	1	33,100	149,000	114,430	305/732

Tabel 17 Parameter yang terukur untuk CBQ IP 192.168.1.2

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	31,7	143	40,628	337/732
2	1	31,1	145	9,851	348/732
3	1	31,5	139	27,832	356/732
Rata-Rata	1	31,433	142,333	26,104	347/732

Tabel 18 Parameter yang terukur untuk CBQ IP 192.168.1.3

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	26,8	131	5,736	434/732
2	1	26,3	136	11,003	427/732
3	1	26,3	137	5,382	426/732
Rata-Rata	1	26,467	134,667	7,374	429/732

Tabel 19 Parameter yang terukur untuk CBQ IP 192.168.1.4

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	30,0	134	43,598	383/732
2	1	29,1	138	25,677	392/732
3	1	29,7	134	47,218	387/732
Rata-Rata	1	29,600	135,333	38,831	387/732

Tabel 20 Parameter yang terukur untuk CBQ IP 192.168.1.5

2. Skenario dengan *leaf class* berdasarkan *port*

a. Skenario dengan satu *leaf class* berdasarkan *port*

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	17,3	498	24,070	0/732
2	1	17,3	498	24,121	0/732
3	1	17,3	498	24,070	0/732
Rata-Rata	1	17,300	498,000	24,087	0/732

Tabel 21 Parameter yang terukur untuk HTB *port* 3001

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	16,7	516	12,680	0/732
2	1	16,7	516	12,718	0/732
3	1	16,7	516	12,714	0/732
Rata-Rata	1	16,700	516,000	12,704	0/732

Tabel 22 Parameter yang terukur untuk CBQ *port* 3001

b. Skenario dengan dua *leaf class* berdasarkan *port*

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	33,9	254	50,785	0/732
2	1	33,8	255	46,579	0/732
3	1	34,0	253	48,698	0/732
Rata-Rata	1	33,900	254,000	48,687	0/732

Tabel 23 Parameter yang terukur untuk HTB *port* 3001

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	34,3	251	30,176	0/732
2	1	34,2	252	29,791	0/732
3	1	34,2	252	29,457	0/732
Rata-Rata	1	34,233	251,667	29,808	0/732

Tabel 24 Parameter yang terukur untuk HTB *port* 3002

Sampel ke -	Besar Pesan	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram

	(Mbyte)				
1	1	31,1	258	10,519	50/732
2	1	31,1	257	11,636	53/732
3	1	31,1	257	11,384	53/732
Rata-Rata	1	31,100	257,333	11,180	52/732

Tabel 25 Parameter yang terukur untuk CBQ *port* 3001

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	32,0	261	63,236	13/732
2	1	32,0	262	54,843	13/732
3	1	32,0	262	55,965	16/732
Rata-Rata	1	32,000	261,667	58,015	14/732

Tabel 26 Parameter yang terukur untuk CBQ *port* 3002

c. Skenario dengan tiga *leaf class* berdasarkan *port*

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	50,3	171	71,950	0/732
2	1	50,4	171	63,113	0/732
3	1	49,9	173	72,074	0/732
Rata-Rata	1	50,200	171,667	69,046	0/732

Tabel 27 Parameter yang terukur untuk HTB *port* 3001

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	51,3	168	49,714	0/732
2	1	51,1	168	56,990	0/732
3	1	51,1	168	52,999	0/732
Rata-Rata	1	51,167	168,000	53,234	0/732

Tabel 28 Parameter yang terukur untuk HTB *port* 3002

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	51,2	168	46,114	0/732
2	1	51,0	169	36,093	0/732
3	1	50,9	169	43,221	0/732
Rata-Rata	1	51,033	168,667	41,809	0/732

Tabel 29 Parameter yang terukur untuk HTB *port* 3003

Sampel ke -	Besar Pesan	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram

	(Mbyte)				
1	1	28,7	180	16,832	292/732
2	1	32,2	197	64,762	179/732
3	1	31,9	188	26,867	222/732
Rata-Rata	1	30,933	188,333	36,154	231/732

Tabel 30 Parameter yang terukur untuk CBQ *port* 3001

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	31,7	173	62,083	261/732
2	1	31,3	179	57,954	246/732
3	1	31,4	175	24,137	258/732
Rata-Rata	1	31,467	175,667	48,058	255/732

Tabel 31 Parameter yang terukur untuk CBQ *port* 3002

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	31,7	196	33,136	205/732
2	1	28,0	180	4,722	303/732
3	1	29,4	175	5,467	293/732
Rata-Rata	1	29,700	183,667	14,442	267/732

Tabel 32 Parameter yang terukur untuk CBQ *port* 3003

d. Skenario dengan empat *leaf class* berdasarkan *port*

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	66,2	130	91,448	0/732
2	1	66,2	130	112,837	0/732
3	1	65,8	131	112,528	0/732
Rata-Rata	1	66,067	130,333	105,604	0/732

Tabel 33 Parameter yang terukur untuk HTB *port* 3001

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	67,7	127	78,033	0/732
2	1	68,0	127	75,319	0/732
3	1	67,6	127	81,595	0/732
Rata-Rata	1	67,767	127,000	78,316	0/732

Tabel 34 Parameter yang terukur untuk HTB *port* 3002

Sampel ke -	Besar Pesan	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram

	(Mbyte)				
1	1	68,0	127	61,701	0/732
2	1	68,1	126	77,475	0/732
3	1	67,7	127	66,752	0/732
Rata-Rata	1	67,933	126,667	68,643	0/732

Tabel 35 Parameter yang terukur untuk HTB *port* 3003

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	67,8	127	50,768	0/732
2	1	68,1	126	58,110	0/732
3	1	67,3	128	45,856	0/732
Rata-Rata	1	67,733	127,000	51,578	0/732

Tabel 36 Parameter yang terukur untuk HTB *port* 3004

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	31,9	159	56,860	283/732
2	1	32,0	152	53,462	311/732
3	1	31,7	152	46,041	323/732
Rata-Rata	1	31,867	154,333	52,121	306/732

Tabel 37 Parameter yang terukur untuk CBQ *port* 3002

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	32,2	137	118,264	356/732
2	1	32,3	133	98,321	365/732
3	1	31,5	144	33,597	341/732
Rata-Rata	1	32,000	138,000	83,394	354/732

Tabel 38 Parameter yang terukur untuk CBQ *port* 3003

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	29,8	132	64,742	397/732
2	1	30,2	133	67,143	385/732
3	1	30,2	132	56,502	388/732
Rata-Rata	1	30,067	132,333	62,796	391/732

Tabel 39 Parameter yang terukur untuk CBQ *port* 3004

Sampel ke -	Besar Pesan	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram

	(Mbyte)				
1	1	26,6	133	12,796	432/732
2	1	27,3	135	20,446	418/732
3	1	27,2	131	25,976	430/732
Rata-Rata	1	27,033	133,000	19,739	427/732

Tabel 40 Parameter yang terukur untuk CBQ *port* 3005

3. Skenario Dengan Parameter Khusus

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	69,1	125	97,004	0/732
2	1	69,0	125	97,344	0/732
3	1	69,0	125	88,895	0/732
Rata-Rata	1	69,033	125,000	94,414	0/732

Tabel 41 Parameter yang terukur untuk HTB IP 192.168.1.2

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	69,0	125	93,661	0/732
2	1	69,0	125	91,640	0/732
3	1	69,1	125	97,171	0/732
Rata-Rata	1	69,033	125,000	94,157	0/732

Tabel 42 Parameter yang terukur untuk HTB IP 192.168.1.3

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	26,8	131	5,736	434/732
2	1	26,3	136	11,003	427/732
3	1	26,3	137	5,382	426/732
Rata-Rata	1	26,467	134,667	7,374	429/732

Tabel 43 Parameter yang terukur untuk CBQ IP 192.168.1.2

Sampel ke -	Besar Pesan (Mbyte)	Waktu pengiriman (detik)	Throughput (kbit/s)	Jitter (milidetik)	Loss Datagram
1	1	30,0	134	43,598	383/732
2	1	29,1	138	25,677	392/732
3	1	29,7	134	47,218	387/732
Rata-Rata	1	29,600	135,333	38,831	387/732

Tabel 44 Parameter yang terukur untuk CBQ IP 192.168.1.3