**Centro Universitário de Brasília – UniCEUB**

**Faculdade de Exatas e Tecnologia - FAET**

**Curso de Engenharia da Computação**

**Fábio Sousa Martins**

# Implementação de um gateway de borda com controle de banda utilizando software livre

**Brasília**

**2005**

**Fábio Sousa Martins**

# Implementação de um gateway de borda com controle de banda utilizando software livre

Trabalho apresentado ao
Centro Universitário de Brasília (UNICEUB)
Como pré-requisito para a obtenção de Certificado de
Conclusão do Curso de
Engenharia da Computação

Orientador: Prof. Mc Francisco Javier de Obaldia Diaz

**Brasília**

**2005**

# Resumo

Esse trabalho propõe duas soluções de gateway de borda com controle de banda utilizando dois softwares livres completamente distintos e fazendo uso de enfileiradores com prioridades de tal forma a permitir que uma rede com tráfego altamente desordenado permita que seu gateway organize os pacotes de forma que o tráfego mais importante seja tratado de forma diferenciada. Desta forma, esse trabalho disponibilizará o conhecimento necessário assim como regras e scripts prontos para a utilização do que há de mais novo na tecnologia de controle de banda por enfileiradores sem ônus algum a quem queira utilizá-la.

Palavras chaves: ALTQ, CBQ, controle de banda, dummynet, enfileiramento, firewall, gateway, iproute2, Linux, OpenBSD, pf, qdisc, QoS, roteador, SFQ, Slackware, software livre, tc, TCP/IP.

# Abstract

This project proposes two border gateway solutions with bandwidth control using completely distinct free software, using queueing disciplines with priorities to allow a network with a completely disordered traffic to have its packets flow by rules, prioritizing more important data. By this, this project will provide the necessary knowledge and also rules and scripts ready to use, using the newest technologies with schedulers without any charging.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Símbolos

| | |
|---|---|
| ALTQ | Alternate Queueing: framework criado para ambientes BSD Unix para controle de filas |
| Backlog | Log de requisições de clientes que ainda não receberam o pacote TCP ACK de retorno |
| Bucket | Buffer utilizado no controle de banda em ambientes Linux |
| Burst | Termo utilizado para designar pequenos tráfegos de rajada |
| CBQ | Class Based Queueing: algoritmo de enfileiramento que permite a subdivisão das suas filas em classes. |
| CB-WFQ | Class Based Weighted Fair Queuing: algoritmo utilizado pela Cisco que agrega funcionalidades do CBQ e do WFQ. |
| CoS | Class of Service: campo utilizado no protocolo IP para designar a classe de serviço. |
| DiffServ | Método de tentar garantir qualidade de serviço em grandes redes. |
| dummynet | Ferramenta utilizada para se testar protocolos e limitar banda no FreeBSD. |
| ECN | Explicity Congestion Notification: algoritmo de notificação de congestionamento em rede. |
| FIFO | Fila simples (First In, First Out) |
| Framework | Estrutura a partir da qual um software pode ser organizado e desenvolvido |
| Handle | Ponteiro utilizado na configuração de controle de banda do Linux para referenciar uma fila. |
| IMQ | Intermediate Queueing device: dispositivo de fila intermediário - dispositivo criado para controlar filas através de políticas utilizando o Netfilter (iptables/ipchains). |
| ipfw | Um dos firewalls utilizados pelo FreeBSD |
| MPU | Minimum Packet Unit: parâmetro utilizado para definir a quantidade de banda que um pacote de tamanho 0 utilizará. |
| MTU | Maximum Transmission Unit: parâmetros que defina o tamanho do datagrama do protocolo. |
| Overlimit | Situação em que são enviados mais pacotes do que uma fila pode processar. |
| Patch | Alteração feita em uma aplicação para consertar bugs ou adicionar funcionalidades. |
| Peakrate | Parâmetro usado para especificar quão rápido o bucket pode ser esvaziado no Linux. |
| PF | Packet Filter: firewall desenvolvido por Daniel Hartmeier para ser usado em ambientes BSD Unix. |
| pfctl | Comando para controle do firewall usado no OpenBSD |
| PFIFO | Algoritmo de fila simples FIFO usada em ambientes Linux, que faz sua contabilização por pacotes. |
| PFIFO_FAST | Algoritmo de fila usado por padrão em ambientes Linux que possui 3 bandas internas. |
| PRIO | Algoritmo de fila de prioridades utilizado no Linux |
| PRIQ | Algoritmo de fila de prioridades utilizada no OpenBSD |
| qdisc | Abreviação para Queueing Discipline utilizada em ambientes baseados em Linux para controlar filas. |
| QoS | Quality of Service: sigla utilizada para designar Qualidade de Serviço |
| RED | Random Early Detection: algoritmo feito para prever um congestionamento de rede antes que ele ocorra. |
| Round-robin | Algoritmo de agendamento de processos que disponibiliza tempos iguais de processamento para cada processo. |
| SFQ | Uma implementação simples de algoritmo fair-queueing que é menos apurado, mas também |

| | |
|---|---|
| | exige menos cálculos. |
| sysctl | Abreviação para System Call: utilizada em ambientes Unix para controlar alguns parâmetros do kernel dinamicamente. |
| TBF | Token Bucket Filter: algoritmo de enfileiramento que limita o tamanho da banda. |
| tc | Traffic Control: arquivo binário utilizado para criar filas no Linux |
| TOS | Type of Service: campo utilizado nos pacotes IP para definição de tipo de serviço. |
| Userspace | Termo utilizado para designar o local onde processos alheios ao kernel estão sendo executados. |
| WFQ | Weighted Fair Queueing: Algoritmo de fila baseado em pesos. |

# Lista de Gráficos

# 1. Introdução

As redes IP constituem um grande universo de milhões de computadores interligados em todo o mundo com uma expectativa constante de crescimento. Essas redes se desenvolveram com facilidade devido ao crescimento da Internet e pela homologação do TCP/IP como protocolo de suporte às aplicações em redes. Não há nenhum aceno em relação à mudanças deste panorama devido à grande massa de computadores usando TCP/IP nas suas comunicações, sejam elas caseiras ou corporativas. Sendo assim, o IP torna-se o padrão universal de suporte para aplicações, pois está presente em milhões de máquinas espalhadas por todo o mundo. As redes TCP/IP foram desenvolvidas com o discurso de que poderiam ser usadas sobre qualquer tipo de meio físico, sendo estes de qualquer tecnologia, apresentando ou não confiabilidade, com alto ou baixo desempenho. Como o TCP/IP é um protocolo simples, ele tem algumas restrições, como por exemplo a falta de garantia no trânsito de pacotes, atrasos etc. Com o crescimento e diversificação das aplicações (voz, multimídia, vídeo-conferência etc.) sobre IP, precisou-se avaliar as limitações do IP, buscando-se alternativas para adequá-lo à nova realidade, pois a banda está compartilhada com um número cada vez maior de usuários podendo ocorrer congestionamento e possíveis perdas de pacotes. Para que se obtenha uma garantia de que o serviço prestado por essas aplicações será realizado com um mínimo de qualidade é preciso aplicar tecnologias que permitam atingir um nível de tráfego satisfatório e confiável para os dados e aplicações. A isso dá-se o nome de qualidade de serviço.

Uma das formas de se atingir uma qualidade de serviço satisfatória é utilizar-se de controle de banda no tráfego TCP/IP.

Esse trabalho mostrará duas formas de se fazer controle de banda em um gateway utilizando um PC de baixo custo e eliminando totalmente os custos com software, já que as duas soluções mostradas utilizarão softwares com o código-aberto. Outra vantagem de se utilizar software de código-aberto é a questão da auditoria do código-fonte que pode ser feita por qualquer pessoa. Todos os tipos de controle e limitação de banda utilizados nesse trabalho chegam até a camada 4 do modelo OSI [Tanembaum, 2003].

O sistema operacional Slackware Linux, escolhido para a primeira solução, é uma das mais tradicionais distribuições de Linux no mercado. O sistema operacional escolhido para a segunda solução, o OpenBSD, é conhecido como o sistema operacional de código-aberto mais seguro do mundo [Lucas, M., 2003].

No capítulo 2, será dada uma visão geral de como atingir bons níveis de QoS com software livre e também serão mostrados alguns produtos comerciais que fazem essa tarefa.

No capítulo 3, será dada uma descrição do projeto aqui proposto, mostrando tanto a topologia utilizada como as duas soluções de controle de banda criadas.

No capítulo 4, será descrito como foi feita a implementação das duas soluções, assim como as configurações que foram criadas e utilizadas nas mesmas. Nesse mesmo capítulo, apresentam-se alguns dos resultados dos testes feitos com as duas máquinas, ressalvando algumas diferenças e resultados encontrados.

No capítulo 5, os resultados encontrados serão comparados, concluindo o projeto e demonstrando como o software livre pode ser utilizado para se fazer QoS nos dias de hoje. Esse capítulo também dá uma visão geral sobre possibilidades de extensão do projeto no futuro.

# 2. Controle de Banda em Redes IP

## 2.1 Definições e Terminologia

Para que se possa entender corretamente o funcionamento de um sistema cujo objetivo é prover um controle de banda é preciso ter de forma clara algumas definições, como: gateway, firewall, QoS (Qualidade de Serviço) e enfileiradores.

### 2.1.1 Gateway

Em uma rede TCP/IP, uma máquina ou host está designada a participar de uma sub-rede, que é definida pela sua sub-máscara de rede. Com a simples configuração de endereço IP e máscara de sub-rede, é possível fazer com que duas máquinas se comuniquem, desde que as duas estejam (além de na mesma sub-rede) no mesmo segmento, físico ou lógico. Dessa forma, se uma máquina dessa mesma sub-rede tentar se comunicar com uma máquina de outra sub-rede, não conseguirá, utilizando-se apenas dessa configuração.

Para resolver esse problema, é preciso definir um "gateway" para aquela máquina [Comer, 1998]. Ou seja, se essa máquina resolve se comunicar com uma outra que não esteja em sua sub-rede, ela deve procurar um caminho alternativo para isso. Dessa forma, a máquina irá requerer a comunicação para o endereço definido pelo seu gateway. E o gateway tratará de tentar encaminhar essa comunicação para a rede ou máquina de destino.

Para se tratar de limitações de banda, é importante a definição do gateway, pois é ele quem estará fazendo o trabalho de controle de tráfego entre as redes em que o mesmo se encontra.

### 2.1.2 Firewall

Um firewall nada mais é do que um hardware ou software que faz o trabalho de tratar o tráfego utilizando-se de filtros para definir o que pode ou não trafegar em determinada interface de rede. Em outras palavras, ele bloqueia ou permite certos tipos de tráfegos pré-definidos por quem o configurou (normalmente o administrador da rede) com o objetivo principal de proteger uma determinada rede de ataques vindos de outras redes [McClure, 2003].

Existem firewalls que agregam mais funcionalidades do que apenas bloquear ou permitir que os pacotes trafeguem por ele. Vários firewalls também tratam o tráfego de forma a ter algum tipo de controle sobre a banda que por ele passa. Portanto, normalmente o trabalho de controle de tráfego está ligado de alguma forma a algum tipo de firewall ou gateway.

### 2.1.3 QoS

A sigla QoS significa "Quality of Service" ou, traduzindo para português, Qualidade de Serviço. É uma sigla muito empregada no mundo da tecnologia da informação, na maioria das vezes com o intuito de aumentar a qualidade em algum dos serviços oferecidos na área de telecomunicações [Jha, 2002].

Dessa forma, a sigla QoS pode ser usada para definir, por exemplo, uma estrutura de alta disponibilidade de meios de telecomunicação. Porém, na maioria das vezes, quando se trata de QoS está se falando de banda de comunicação. QoS normalmente é visto como algum tipo de controle na quantidade de tráfego de dados que passam por um link de comunicação e em como prover a qualidade necessária para que aquele link tenha a banda necessária para executar os serviços que nele trafegam.

### 2.1.4 Enfileiradores

Enfileirar algo é armazená-lo em algum lugar, de maneira organizada, enquanto aguarda processamento. Numa rede de computadores, quando pacotes de dados são enviados por um host, eles entram numa fila onde aguardam processamento pelo sistema operacional. O sistema operacional decide qual fila e quais pacotes nesta fila devem ser processados. A ordem em que o sistema operacional escolhe os pacotes a processar pode afetar a performance da rede.

Como regra geral, quando um pacote chega a um gateway ou um roteador, ele entra em uma fila para ser processado. Caso não haja nenhum tipo de regra para tratar esse tráfego, essa fila será uma fila FIFO (First In, First Out) simples. Ou seja, os pacotes vão chegando, e vão sendo processados conforme a ordem de chegada.

Com o uso de enfileiradores, pode-se alterar esse comportamento normal dos roteadores e gateways e mudar a forma como os pacotes são tratados. E possível tanto limitar a velocidade com que os dados saem de uma interface, como também priorizar certos tipos de tráfego para que sejam processados antes de outros de menor importância. Dessa forma, é possível determinar como os pacotes são enviados. A esse processo, dá-se o nome de enfileiramento [Jha, 2002].

## 2.2 QoS em TCP/IP

Do modo como a internet trabalha, não é possível como ter controle do que outras pessoas nos enviam. É como uma caixa de correio em casa: não há como influenciar o mundo sobre a quantidade de correspondência que nos mandam, a não ser que se consiga entrar em contato com todos.

Como a internet é baseada em TCP/IP, existem alguns recursos que podem ser utilizados. O TCP/IP não tem um modo de saber qual é a velocidade da rede entre duas estações, então ele inicia a conexão de forma mais rápida [RFC2001, 1997] e, quando os pacotes começam a se perder, pois não há mais como processá-los devido à limitação de um meio físico, por exemplo, a velocidade de envio é diminuída.

Dessa forma, existem algumas maneiras de controlar a velocidade de saída de um link.

### 2.2.1 QoS em Linux

#### 2.2.1.1 Disciplinas de Enfileiramento para Controle de Banda

O Linux utiliza, como estrutura para prover qualidade de serviço no seu kernel [Maxwell, 2000], alguns sistemas de filas, que devem ser habilitados no kernel do sistema para o seu correto funcionamento [Hubert, 2003].

Para poder tratar o tráfego e encaminhá-lo corretamente para as filas, assim como para defini-las, criar subfilas e gerenciá-las, utiliza-se o comando “tc” [Linux Man, 2001] , contido na suíte de aplicativos iproute2 [Lamb, 1999].

Para evitar confusões, o comando “tc” utiliza as seguintes regras para especificação de banda.

mbps = 1024 kbps = 1024 * 1024 bps => byte/s
mbit = 1024 kbit => kilo bit/s.
mb = 1024 kb = 1024 * 1024 b => byte
mbit = 1024 kbit => kilo bit.

Internamente, o número é armazenado em bps.

Quando o comando tc mostra uma taxa, ele utiliza o seguinte:

1Mbit = 1024 Kbit = 1024 * 1024 bps => byte/s

*Disciplinas de filas*

Através do enfileiramento, pode-se determinar a forma como os dados são enviados, ou seja, através do enfileiramento não se pode limitar a capacidade de entrada (afinal, se o tráfego já chegou naquela interface, não há mais limitação nenhuma a fazer). Dessa forma, pode-se limitar a velocidade como os dados são transmitidos.

Suponha que se tenha um roteador e se pretenda prevenir que alguns hosts da rede façam downloads muitos rápidos. Para isso, é necessário limitar a banda na interface interna do roteador (que é a que envia dados para a rede interna).

É necessário também ter certeza que se está controlando o gargalo do link. Por exemplo: tendo-se uma interface ethernet configurada a uma velocidade de 100 Mbps e um roteador que está configurado a 256 Kbps, é necessário que se tenha certeza de que não serão enviados mais pacotes do que o roteador pode processar. De outra forma, será o roteador que estará controlando o link e limitando a capacidade da banda. Em outras palavras, para se ter a gerência da fila, é preciso ter conhecimento sobre o link mais lento da rede.

Dessa forma, tem-se várias formas de se controlar o enfileiramento. Essas formas são chamadas disciplinas e são subdivididas em disciplinas de enfileiramento simples e disciplinas de enfileiramento de classe.

2.2.1.2 Disciplinas de enfileiramento simples

O Linux utiliza “qdiscs” para fazer seu tratamento de tráfego por enfileiramento. Qdisc é uma abreviação para “Queueing Disciplines”, que em português, significa disciplinas de enfileiramentos.

Utilizando qdiscs pode-se alterar a forma como os dados são enviados. Qdiscs simples são aquelas que apenas reagendam, adicionam atraso ou descartam pacotes.

O Linux possui suporte a vários tipos de filas diferentes, que serão explicadas abaixo. O tipo de disciplina de enfileiramento mais usado é o PFIFO_FAST, que é a fila padrão do Linux.

*PFIFO_FAST*

Essa fila é uma fila do tipo FIFO (First In, First Out), o que significa que nenhum pacote recebe tratamento especial. Essa fila é subdividida em três “bandas”. Em cada uma delas, a regra FIFO se aplica. Contudo, se houver pacotes aguardando para serem processados na banda 0, a banda 1 não será processada. O mesmo acontece para a banda 2, em relação a banda 1.

O kernel respeita a flag TOS (Type Of Service) [RFC1349, 1992] do campo TCP/IP, e insere pacotes com “mínimo atraso” (minimum delay) na banda 0.

Como a fila PFIFO_FAST é a fila padrão, não é necessário fazer nenhuma configuração adicional no sistema. Porém, existem alguns parâmetros, a seguir descritos, que podem ser configurados.

Priomap:

O priomap determina como as prioridades de pacotes se mapeiam às bandas. Esse mapeamento ocorre baseado no octeto TOS do pacote:

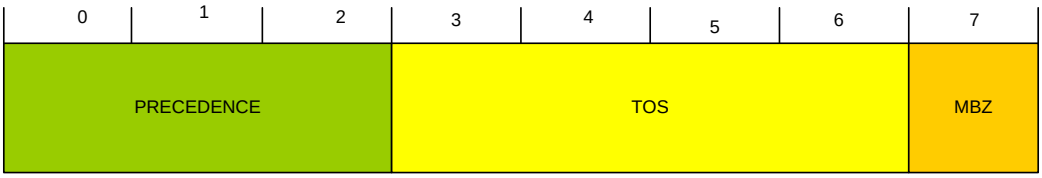


**Figura 2.1 – Octeto TOS do pacote [RFC1349, 1992]**

Os 4 bits TOS (campo TOS) são definidos como:

| Binário | Decimal | Significado |
|---|---|---|
| 1000 | 8 | Minimize delay (md) |
| 100 | 4 | Maximize throughput (mt) |
| 10 | 2 | Maximize reliability (mr) |
| 1 | 1 | Minimize monetary cost (mmc) |
| 0 | 0 | Normal Service |

**Tabela 2.1 – Configuração para os campos TOS [RFC1349, 1992]**

Como existe 1 bit à direita desses 4 bits, o valor correto do campo TOS é o dobro do valor dos bits TOS. Se esses valores forem vistos em um analisador de protocolos qualquer (“tcpdump”, por exemplo), serão mostrados conforme mostra o primeiro campo da tabela abaixo:

| TOS | Bits | Means | Linux Priority | | Band |
|---|---|---|---|---|---|
| 0x0 | 0 | Normal Service | 0 | Best Effort | 1 |
| 0x2 | 1 | Minimize Monetary Cost | 1 | Filler | 2 |
| 0x4 | 2 | Maximize Reliability | 0 | Best Effort | 1 |
| 0x6 | 3 | mmc+mr | 0 | Best Effort | 1 |
| 0x8 | 4 | Maximize Throughput | 2 | Bulk | 2 |
| 0xa | 5 | mmc+mt | 2 | Bulk | 2 |
| 0xc | 6 | mr+mt | 2 | Bulk | 2 |
| 0xe | 7 | mmc+mr+mt | 2 | Bulk | 2 |
| 0x10 | 8 | Minimize Delay | 6 | Interactive | 0 |
| 0x12 | 9 | mmc+md | 6 | Interactive | 0 |
| 0x14 | 10 | mr+md | 6 | Interactive | 0 |
| 0x16 | 11 | mmc+mr+md | 6 | Interactive | 0 |
| 0x18 | 12 | mt+md | 4 | Int. Bulk | 1 |
| 0x1a | 13 | mmc+mt+md | 4 | Int. Bulk | 1 |
| 0x1c | 14 | mr+mt+md | 4 | Int. Bulk | 1 |
| 0x1e | 15 | mmc+mr+mt+md | 4 | Int. Bulk | 1 |

**Tabela 2.2 – Significado dos campos TOS [RFC1349, 1992]**

A segunda coluna contém o valor dos 4 bits relevantes do campo TOS, seguido do significado. Por exemplo, 15 representa um pacote desejando Minimal Monetary Cost, Maximum Reliability, Maximum Throughput e Minimum Delay.

A quarta coluna lista os modos como o kernel do Linux interpreta os bits do campo TOS, mostrando a qual prioridade ele está mapeado.

A última coluna mostra o resultado do priomap padrão. Na linha de comando, o priomap padrão se parece com isso:

1, 2, 2, 2, 1, 2, 0, 0 , 1, 1, 1, 1, 1, 1, 1, 1

Isso significa que prioridade 4, por exemplo, é mapeada à banda número 1. O priomap também permite que se listem prioridades maiores (>7) que não correspondem aos mapeamentos do campo TOS, mas que são definidos por outros meios.

| Type-of-Service Value | | |
|---|---|---|
| Protocol | TOS Value | Meaning |
| | | |
| TELNET (1) | 1000 | (minimize delay) |
| | | |
| FTP | | |
| Control | 1000 | (minimize delay) |
| Data (2) | 100 | (maximize throughput) |
| | | |
| TFTP | 1000 | (minimize delay) |
| | | |
| SMTP (3) | | |
| Command phase | 1000 | (minimize delay) |
| DATA phase | 100 | (maximize throughput) |
| | | |
| Domain Name Service | | |
| UDP Query | 1000 | (minimize delay) |
| TCP Query | 0 | |
| Zone Transfer | 100 | (maximize throughput) |
| | | |
| NNTP | 1 | (minimize monetary cost) |
| | | |
| ICMP | | |
| Errors | 0 | |
| Requests | 0000 (4) | |
| Responses | <same as request> (4) | |
| | | |
| Any IGP | 10 | (maximize reliability) |
| | | |
| EGP | 0 | |
| | | |
| SNMP | 10 | (maximize reliability) |
| | | |
| BOOTP | 0 | |

**Tabela 2.3 - Valores de TOS [RFC 1349, 1992]**

Txqueuelen

Esse parâmetro representa o tamanho da fila, que é retirado da configuração de sua interface, o que pode ser visto e definido com os comandos ip e ifconfig. Para definir um tamanho de fila 10, é só executar o comando: ifconfig eth0 txqueuelen 10

*Token Bucket Filter*

O Token Bucket Filter (TBF) é uma disciplina de enfileiramento simples onde apenas trafegam pacotes que chegam a uma determinada taxa, a qual não excede uma taxa definida administrativamente, mas com a possibilidade de permitir pequenos tráfegos de rajada excedendo essa taxa.

A fila TBF é muito precisa e não afeta muito a rede, além de não exigir muito do processador. Normalmente é a primeira escolha quando o intuito é apenas diminuir a velocidade de uma interface.

A implementação da fila TBF consiste em um buffer (bucket) que é constantemente "enchido" de alguns pedaços virtuais de informação, chamados de *tokens*, em uma taxa de velocidade específica (token rate). O parâmetro mais importante do bucket é o seu tamanho, que é o número de *tokens* que ele pode armazenar.

Cada *token* que chega, coleta um pacote que chega da fila de dados e é, então, deletado do bucket. Associando esse algoritmo aos dois fluxos – *token* e dados -, tem-se 3 cenários possíveis:

1) Os dados chegam na fila TBF em uma taxa que é igual à taxa de *tokens* que chegam. Nesse caso, o pacote que chega tem um *token* correspondente e passa pela fila sem nenhum atraso;
2) Os dados chegam na fila TBF em uma taxa que é menor do que a taxa do *token*. Somente uma parte dos *tokens* se acumulam, até o tamanho do bucket. Os *tokens* não utilizados podem então ser usados para enviar dados na velocidade que está excedendo o tamanho padrão do *token*, no caso de pequenas rajadas ocorrerem;
3) Os dados chegam na fila TBF em uma taxa maior do que a taxa do *token*. Isso significa que o bucket brevemente irá ficar sem *tokens*, o que causará a fila TBF a se regular por um tempo. Isso é chamado de "overlimit situation". Se os pacotes continuarem chegando, os pacotes começarão a ser descartados.

O último cenário é muito importante porque permite que se molde administrativamente a banda disponível que está passando pelo filtro.

A acumulação de *tokens* permite que uma pequena rajada de dados acima do limite continue passando sem perda, mas se os pacotes continuarem a passar acima do limite, os dados irão constantemente ganhar atrasos e depois ser descartados.

Na atual implementação do algoritmo TBF, os *tokens* correspondem a bytes e não a pacotes.

Alguns parâmetros que podem ser alterados:

Limite (latency):

Limite é o número de bytes que podem ser enfileirados aguardando que os *tokens* se tornem disponíveis. É possível também especificar esse parâmetro definindo o valor da latência, o que especifica o máximo de tempo que um pacote pode estar na fila TBF. Os cálculos levam em conta o tamanho do bucket, a taxa e o peakrate (se definido)

Burst/Buffer/Maxburst

É o tamanho do bucket, em bytes. Esse é o máximo de bytes que os *tokens* têm disponível instantaneamente. No geral, taxas de controle mais altas requerem um buffer maior.

Se o buffer for pequeno demais, pacotes podem ser descartados pois chegam mais *tokens* por unidade de tempo do que cabem no bucket.

MPU

Um pacote de tamanho zero não utiliza zero de banda. Para o caso do padrão Ethernet, nenhum pacote utiliza menos do que 64 bytes. O campo Minimum Packet Unit determina o uso mínimo de *tokens* para um pacote.

Rate

A velocidade definida para se limitar a banda.

Se o bucket tiver *tokens* e for permitido que o mesmo se esvazie, por padrão, ele o faz em uma velocidade infinita. Se isso for inaceitável, é necessário utilizar os seguintes parâmetros:

Peakrate

Se os *tokens* estão disponíveis e os pacotes chegam, por padrão, eles são enviados imediatamente. Pode ser que não seja esse o objetivo, especialmente caso se esteja trabalhando com um bucket largo.

O peakrate pode ser usado para especificar o quão rápido é permitido que o bucket seja esvaziado. Isso pode ser alcançado soltando-se um pacote, e depois aguardando tempo suficiente, e depois soltando outro. As esperas são calculadas e depois os pacotes são enviados no valor do peakrate.

Contudo, por causa da resolução de timer de 10ms padrão do Unix, com 10.000 bits de pacotes em média, o sistema fica limitado a 1 mbit/s de peakrate.

MTU/Minburst

O peakrate de 1 Mbit/s não é muito útil se a taxa regular que estiver sendo usada for maior do que isso. Um peakrate maior é possível enviando mais pacotes por unidade de tempo, o que efetivamente significa que pode-se criar um segundo bucket. Esse segundo bucket, por padrão, é um simples pacote.

Para calcular o peakrate máximo possível, basta multiplicar o MTU [Stevens, 1993] configurado por 100.

Exemplo de Configuração

Uma configuração simples, porém muito útil é essa:

```
# tc qdisc add dev ppp0 root tbf rate 220kbit latency 50ms burst 1540
```

A linha acima diminui a velocidade de envio para uma taxa (no caso, 220 Kbit/s) que não envia para uma fila em um modem – a fila ficará no Linux, onde pode ser controlada a um tamanho limitado.

Essa linha de comando cria uma qdisc raiz no dispositivo ppp0 utilizando o algoritmo TBF com uma taxa de 220 Kbits, latência de 50ms e um bucket de 1540 bytes.

Tendo-se um periférico de rede com uma fila grande, como um modem DSL ou um Cable Modem, e o mesmo estiver ligado a um dispositivo rápido, como uma interface Ethernet, fazer um upload acaba com a interatividade.

Isso é porque fazer um upload irá encher a fila do modem, que provavelmente é grande, porque ajuda a alcançar um bom throughput no upload. Mas esse não é o objetivo. O objetivo é que a fila não fique muito grande, para que a interatividade permaneça e seja possível ainda utilizar-se da banda para outros tipos de tráfego enquanto se envia dados.

*Stochastic Fairness Queueing*

SFQ é uma implementação simples do algoritmo da família “fair queueing”. É menos apurado do que outros, mas também requer menos cálculos, enquanto permanece quase que perfeitamente justo.

A palavra-chave na implementação do SFQ é conversação (ou fluxo), o que corresponde a uma sessão TCP ou um stream UDP. O tráfego é dividido em um grande número de filas FIFO, uma para cada conversação. O tráfego é então enviado no modo “round-robin”, permitindo a cada sessão que envie dados.

Isso leva a um comportamento extremamente justo e proíbe que uma simples conversação tome conta do link e interfira em outras conversações. O SFQ é também chamado de estocástico porque ele na verdade não aloca uma fila para cada sessão, ele tem um algoritmo que divide o tráfego em um número limitado de filas usando um algoritmo de hash.

Por causa do hash, múltiplas sessões podem terminar no mesmo bucket, o que irá dividir cada chance de cada sessão enviar um pacote, dividindo a velocidade efetiva disponível. Para impedir que essa situação se torne perceptível ao usuário final, a fila SFQ troca seus algoritmos de hash com freqüência para que quaisquer duas sessões que colidam só o façam por um pequeno espaço de tempo (alguns segundos).

É importante notar que a fila SFQ só é útil no caso da interface de saída estar realmente cheia. Se não estiver, então não haverá filas na máquina Linux, o que não provocará efeito algum.

Especificamente, definir o SFQ em uma interface Ethernet ligada diretamente a um cable modem ou roteador ADSL não é de muita utilidade se não for para combiná-la com outras formas de fila.

Existem alguns parâmetros a serem configurados no SFQ:

Perturb

Reconfigura o hashing na quantidade de segundos definido por esse parâmetro. Se não for definido, o hash não será reconfigurado nunca.

Quantum

Quantidade de bytes que um stream é permitido desenfileirar antes que a próxima fila seja processada. O padrão é 1 maximum sized packet (MTU-sized). Não se deve definir um valor menor que o MTU.

Limit

O número total de pacotes que podem ser enfileirados por essa fila SFQ (antes desta começar a descartá-los).

Exemplo de Configuração

Tendo-se um dispositivo que tem uma velocidade de link idêntica à banda disponível, como um modem, essa configuração ajudará a ter um comportamento justo da banda:

```
# tc qdisc add dev ppp0 root sfq perturb 10
# tc -s -d qdisc ls
qdisc sfq 800c: dev ppp0 quantum 1514b limit 128p flows 128/1024 perturb 10sec
 Sent 4812 bytes 62 pkts (dropped 0, overlimits 0)
```

O número 800c é automaticamente designado

Dessa forma, tem-se a PFIFO_FAST, a TBF e a SFQ, e cada uma delas tem seu uso apropriado.

2.2.1.3 Quando usar cada fila

Resumindo, existem as filas simples que gerenciam o tráfego reordenando, diminuindo a velocidade ou descartando pacotes. As filas simples são usadas quando não se quer utilizar subdivisões dentro da mesma banda.

Para utilizar uma fila apenas para diminuir a velocidade do tráfego de saída, a solução mais simples é utilizar o TBF. Funciona para bandas largas, desde que se escale o bucket.

Se o link estiver realmente cheio e se desejar ter certeza de que nenhuma sessão simples domine a banda de saída, o ideal é utilizar o SFQ.

Para limitar a tráfego de entrada que não está sendo redirecionado, o ideal é utilizar o limitador Ingress Policer que na verdade não é um limitador e sim uma Política de Tráfego.

Se o caso for redirecionar o tráfego, o ideal é utilizar uma fila TBF na interface em que se está redirecionando os dados. Isso só não é recomendado quando se está querendo limitar o tráfego de saída em diversas interfaces, pois nesse caso, o único fator comum é a interface de entrada. Nesse caso, o ideal é utilizar o Ingress Policer.

Se o objetivo não é limitar, mas apenas ver qual a carga que se tem em uma interface, o ideal é utilizar a fila PFIFO (e não a PFIFO_FAST). Ela é deficiente em banda interna e trabalha como uma fila FIFO, mas contabiliza o tamanho do backlog.

2.2.1.4 Terminologia utilizada no Linux para se tratar com qdiscs

Para entender as configurações mais complicadas, é necessário explicar alguns conceitos primeiro. Por causa da complexidade e do assunto ser algo relativamente recente, às vezes palavras diferentes são usadas para dizerem a mesma coisa [Hubert, 2003].

*Disciplinas de filas (qdisc)*

Um algoritmo que gerencia a fila de um dispositivo, tanto no sentindo de entrada (ingress) como no sentido de saída (egress).

root qdisc

O “root qdisc” é a disciplina de fila que está anexada ao dispositivo.

Classless qdisc

É uma disciplina de fila que não possui configuração para subdivisão interna.

Classful qdisc

Uma disciplina de fila de classe (ou classful) é uma disciplina de fila que contém múltiplas classes. Algumas dessas classes contêm uma outra disciplina de classes interna a ela, que pode ser mais uma vez uma disciplina de fila de classe, ou uma disciplina de fila simples. De acordo com a definição estrita, PFIFO_FAST é uma disciplina de filas de classe porque contém 3 bandas, que são, de fato, classes. Contudo, do ponto de vista do usuário, é uma disciplina de fila simples, pois as classes não podem ser controladas através da linha de comando do “tc”.

Classes

Uma disciplina de filas de classe pode ter muitas classes, cada uma delas internas à disciplina de fila principal. Uma classe, em contrapartida, pode ter várias outras classes adicionadas a ela. Portanto, uma classe pode

ter uma disciplina de fila como pai ou uma outra classe como pai, em uma hierarquia de árvore. Uma classe folha é uma classe que não possui classes filhas. Essa classe tem uma disciplina de fila anexada a ela. Essa disciplina de fila é responsável por enviar os dados para aquela classe. Quando se cria uma classe, uma disciplina de fila FIFO é anexada a ela. Quando se adiciona uma classe filha, a disciplina de fila é removida. Para as classes de folha, uma disciplina de filas FIFO é anexada a ela. Quando se adiciona uma classe filha, essa disciplina de fila é removida. Para as classes folha, essa disciplina de fila pode ser trocada por uma outra disciplina de fila mais adequada. Pode-se até trocar essa disciplina de filas FIFO por uma disciplina de filas de classe e então adicionar uma classe extra.

Classifier (classificador)

Cada disciplina de filas de classe precisa determinar para qual classe ela deve enviar um pacote. Isso é feito utilizando o classificador.

Filter

A classificação é feita utilizando filtros. Um filtro contém um número de condições que se forem verdadeiras, fazem um filter match (uma correspondência de filtro).

Scheduling (agendamento)

Uma disciplina de fila pode, com a ajuda de um classificador, decidir que alguns pacotes precisam sair antes de outros. Esse processo é chamado de agendamento, e é executado, por exemplo, pela disciplina de fila PFIFO_FAST, mencionada anteriormente. O agendamento também é chamado de reordenação.

Shaping

É o processo de gerar atraso em pacotes antes de eles saírem para confirmar que o tráfego irá sair a uma determinada taxa. O processo de limitação ocorre no egress (tráfego de saída). Coloquialmente, descartar pacotes para diminuir o tráfego também é chamado freqüentemente de limitação.

Policing

Gerar atraso ou descartar pacotes para fazer com que o tráfego permaneça abaixo de uma banda configurada. No Linux, o uso de políticas podem apenas descartar pacotes e não gerar atraso no mesmo – não há fila “ingress” (tráfego de entrada).

Work-Conserving

Uma disciplina de filas conservativa sempre entrega um pacote se ele estiver disponível. Em outras palavras, ele nunca atrasa o pacote se o adaptador de rede estiver pronto para enviar um (no caso de uma disciplina de filas egress).

Non-Work-Conserving

Algumas filas, como o exemplo do Token Bucket Filter, podem precisar reter um pacote por um certo tempo para limitar a banda. Isso significa que elas algumas vezes se recusam a passar um determinado pacote, mesmo que possuam um pacote disponível. É chamado de disciplina de fila não-conservativa.

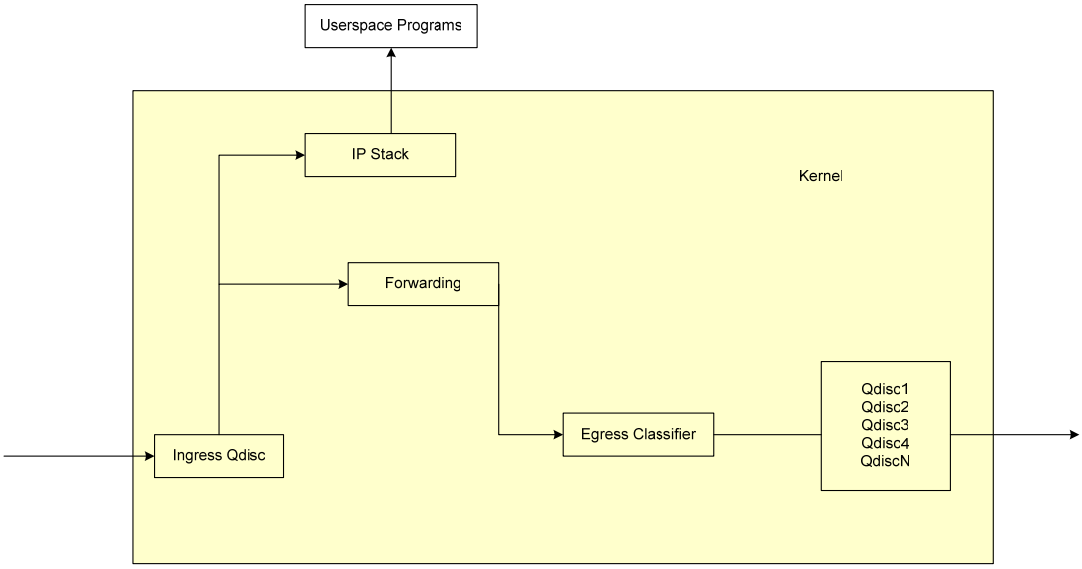


**Figura 2.2 – O enfileiramento no kernel**

Na Figura 2.2, o quadro grande representa o kernel. A seta mais a esquerda representa o tráfego entrando na máquina vindo da rede. Ele então é alimentado para a disciplina de filas de entrada (ingress) que pode aplicar filtros a um pacote e decidir descartá-lo. Isso é chamado “Policing”.

Isso ocorre no início do processamento de tráfego pelo kernel, e portanto, é um lugar muito bom para se descartar tráfego, sem consumir muita CPU.

Se for permitido ao pacote continuar, ele pode ser destinado a uma aplicação local, o que nesse caso entra para a pilha IP para ser processado, e entregue a um programa no userspace. O pacote pode também ser redirecionado sem entrar em nenhuma aplicação, que nesse caso é destinado para o egress. Programas userspace podem também enviar dados, que é então examinado e redirecionado para o classificador de saída (egress). Lá ele será investigado e enfileirado para alguma das disciplinas de filas. No caso de não haver configuração, existe apenas uma disciplina de filas instaladas, que é o PFIFO_FAST, que sempre recebe o pacote.

O pacote agora está em uma fila, aguardando que o kernel requeira que o pacote seja transmitido pela interface de rede. Isso é chamado desenfileiramento.

Isso também acontece no caso de só haver um adaptador de rede – a seta entrando e saindo do kernel não deve ser tomada de forma muito literal. Cada adaptador de rede tem tanto a parte de ingress como a de egress.

2.2.1.5 Classful Queueing Disciplines

Disciplinas de fila de classe (classful) são muito úteis quando se tem diferentes tipos de tráfego que devem ter diferentes tipos de tratamento. Uma das disciplinas de fila de classe é chamada CBQ (class-based queueing) [Linux Man, 2001] e é tão amplamente mencionada que as pessoas identificam o enfileiramento por classe apenas como sendo CBQ, o que não é verdade.

O CBQ é apenas uma das filas mais antigas – e talvez a mais complexa delas. Existe um mito entre usuários de Unix, chamado de “sendmail effect”, que ensina que qualquer tecnologia complexa que não venha com documentação, deve ser a melhor disponível. Isso enquadraria o CBQ, pois nem sempre consegue-se os objetivos de controle que se quer apenas usando CBQ.

*Fluxo com disciplinas de filas de classe*

Quando o tráfego entra em uma fila de classe, ele precisa ser enviado para uma das classes internas – ele precisa ser classificado. Para determinar o que precisa ser feito com um pacote, os chamados filtros são consultados. É importante saber que os filtros são chamados de uma fila de classe, e não o contrário.

Os filtros anexados a cada disciplina de classe retornam uma decisão, e a fila utiliza-se disso para enfileirar o pacote a uma das classes. Cada subclasse pode tentar outros filtros para ver se outras instruções se aplicam. Se não, a classe enfileira o pacote para a fila que o contém.

Além de conter outras filas, a maioria das filas de classe também faz limitação. Isso é útil para fazer tanto o agendamento de pacotes (com SFQ, por exemplo) e o controle da taxa. Isso é necessário nos casos onde se tem uma interface de alta velocidade (por exemplo, ethernet) conectada a uma interface de menor velocidade (um Cable Modem, por exemplo).

Se for utilizado somente SFQ, nada irá acontecer, já que os pacotes entram e saem do roteador sem atraso algum: a interface de saída é mais rápida do que o link atual. Não há fila para agendar neles.

*Família de qdiscs: roots, handles, siblings e pais*

Cada interface tem uma disciplina de fila de saída. Por padrão, ela é a (já mencionada anteriormente) fila simples PFIFO_FAST. A cada disciplina de fila e cada classe é designado um handle, que pode ser usado para configuração posterior para se referir aquela disciplina de fila. Além da qdisc de saída, uma interface pode também ter qdiscs de entrada, o que policia o tráfego que entra.

Os handles das qdiscs consistem em duas partes: um número principal e um número secundário (<major>:<minor>). Isso é normalmente usado para nomear a qdisc raiz “1”, que é igual a “1:0”. O número secundário de uma qdisc é sempre zero.

As classes precisam ter o mesmo número principal que seus pais. Esse número principal precisa ser único em uma configuração de entrada ou de saída. O número secundário também precisa ser único entre uma qdisc e sua classe.

*Uso dos filtros para classificar o tráfego*

Uma hierarquia típica seria algo como a árvore abaixo:

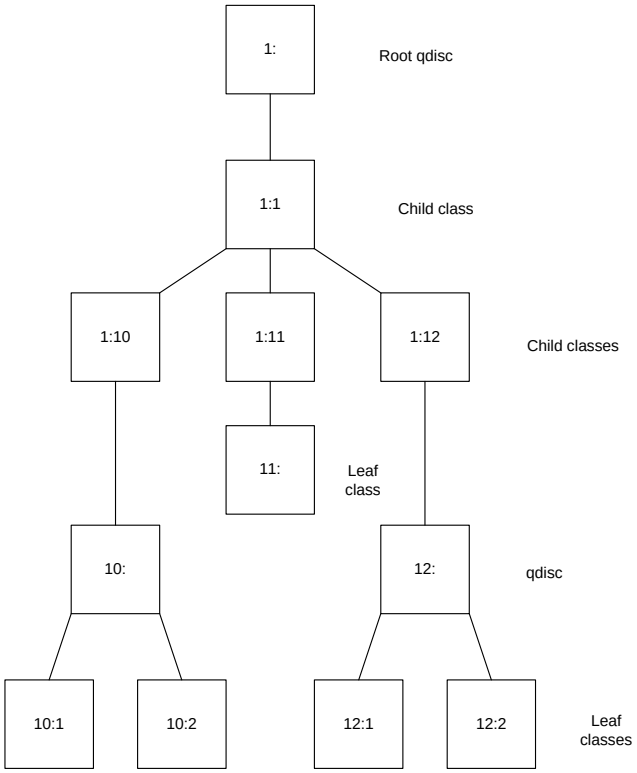


**Figura 2.3 - Exemplo de hierarquia de filas**

Os pacotes são enfileirados e desenfileirados na qdisc raiz. O kernel comunica-se apenas com a qdisc raiz, nunca com as suas subclasses diretamente.

Um pacote pode ser classificado em uma cadeia como essa:

1: -> 1:1 -> 1:12 -> 12: -> 12:2

O pacote agora reside em uma fila em uma qdisc anexada a classe 12:2. Nesse exemplo, um filtro foi anexado a cada nó da árvore, cada um escolhendo um ramo.

1: -> 12:2

Nesse caso, um filtro anexado à raiz decidiu enviar o pacote diretamente a 12:2

*Desenfileiramento dos pacotes para o hardware*

Quando o kernel decide que ele precisa extrair pacotes para enviar para a interface, a qdisc raiz 1 recebe uma requisição de desenfileiramento, que é passada para o 1:1 que é então passada para o 10:, 11: e 12: e cada um deles passa a seus filhos. Nesse caso, o kernel precisa caminhar toda a árvore, porque apenas o 12:2 contém um pacote.

Resumindo, classes só podem falar com seu qdisc pai, nunca com uma interface. Somente a qdisc raiz é desenfileirada pelo kernel.

O resultado disso é que as classes nunca são desenfileiradas mais rápidas do que seus pais permitirem. Desse modo, pode-se ter filas SFQ numa classe interna, o que não faz nenhuma limitação, apenas reagendamento, e ter uma fila limitadora numa qdisc externa, que irá fazer a limitação.

*A qdisc PRIO*

A qdisc PRIO não limita, ela apenas subdivide o tráfego baseada em como estão configurados seus filtros. A qdisc PRIO pode ser considerada um tipo de PFIFO_FAST mais apurada, onde cada banda é uma classe separada ao invés de uma simples fila FIFO.

Quando um pacote é enfileirado para a qdisc PRIO, uma classe é escolhida baseada nos comandos de filtro que tiverem sido dados. Por padrão, 3 classes são criadas. Essas classes contêm qdiscs FIFO puras sem estrutura interna, mas é possível trocar essas por qualquer qdisc que se tenha disponível.

Quando um pacote precisa ser desenfileirado, a classe :1 é tentada primeiro. Classes mais altas são somente usadas se nenhuma das bandas menores for utilizada para enviar um pacote.

Essa qdisc é muito útil no caso de se querer priorizar alguns tipos de tráfego sem utilizar somente as flags TOS, mas usando todas as funcionalidades dos filtros do comando “tc”. Pode-se também adicionar outra qdisc para uma das 3 classes pré-definidas, enquanto PFIFO_FAST é limitada à qdiscs FIFO simples.

A qdisc PRIO deve ser usada apenas se o link estiver realmente cheio (assim como a qdisc SFQ), ou ela deve ser inserida dentro de uma qdisc de classe que não limite banda. Essa última vale para quase todos os dispositivos tipo Cable Modem ou modems ADSL.

Formalmente, a qdisc PRIO é um agendador conservativo.

Parâmetros

Os seguintes parâmetros são reconhecidos pelo comando “tc”.

Bands

Número de bandas a criar. Cada banda é de fato uma classe. Se esse número for trocado, alguns parâmetros também devem ser trocados:

Priomap

Se os filtros “tc” não forem utilizados para classificar o tráfego, a qdisc PRIO procura no campo TC_PRIO de prioridade para decidir como enfileirar o tráfego.

Isso funciona perfeitamente com a qdisc PFIFO_FAST, mencionada antes.

As bandas são classes, e são chamadas: principal:1, principal:2 e principal:3 por padrão, então se a qdisc PRIO é chamada 12, o filtro “tc” deve ser aplicado para 12:1 para conceder uma prioridade maior.

A banda 0 vai para o número secundário 1. A banda 1 para o número secundário 2, e assim por diante.

Exemplo de configuração:

Criando-se a árvore abaixo como exemplo:

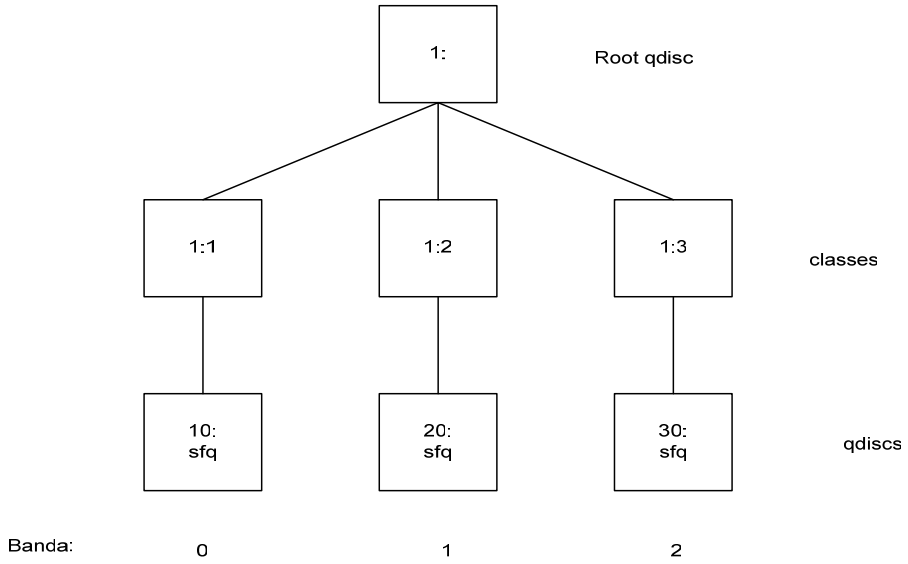


**Figura 2.4 - Exemplo de qdisc PRIO**

Na Figura 2.4, o tráfego de menor prioridade vai para 30, tráfego interativo para 20: ou 10:

Linhas de comando:

```
# tc qdisc add dev eth0 root handle 1: prio
```

Isso cria instantaneamente as classes 1:1, 1:2 e 1:3.

```
# tc qdisc add dev eth0 parent 1:1 handle 10: sfq
# tc qdisc add dev eth0 parent 1:2 handle 20: tbf rate 20kbit buffer 1600 limit 3000
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
```

Ou seja, o que foi criado foi:

```
# tc -s qdisc ls dev eth0
qdisc sfq 30: quantum 1514b
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)
qdisc tbf 20: rate 20Kbit burst 1599b lat 667.6ms
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)
qdisc sfq 10: quantum 1514b
 Sent 132 bytes 2 pkts (dropped 0, overlimits 0)
qdisc prio 1: bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 174 bytes 3 pkts (dropped 0, overlimits 0)
```

Ou seja, a banda 0 já tem algum tráfego, e o pacote foi enviado enquanto o comando era executado.

Fazendo algumas transferências de dados com uma ferramenta que defina os flags TOS, tem-se o que segue:

```
# scp tc fabio@192.168.1.3:./
fabio@192.168.1.3's password:
tc                100% |*****************************|   353 KB    00:00
# tc -s qdisc ls dev eth0
qdisc sfq 30: quantum 1514b
 Sent 384228 bytes 274 pkts (dropped 0, overlimits 0)
qdisc tbf 20: rate 20Kbit burst 1599b lat 667.6ms
 Sent 2640 bytes 20 pkts (dropped 0, overlimits 0)
qdisc sfq 10: quantum 1514b
 Sent 2230 bytes 31 pkts (dropped 0, overlimits 0)
qdisc prio 1: bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 389140 bytes 326 pkts (dropped 0, overlimits 0)
```

Todo o tráfego foi para o handle 30, o que é a banda de menor prioridade, como o intencionado. Agora, para verificar que o tráfego interativo passa em bandas mais altas, basta criar alguns tráfegos:

```
# tc -s qdisc ls dev eth0
qdisc sfq 30: quantum 1514b
 Sent 384228 bytes 274 pkts (dropped 0, overlimits 0)
qdisc tbf 20: rate 20Kbit burst 1599b lat 667.6ms
 Sent 2640 bytes 20 pkts (dropped 0, overlimits 0)
qdisc sfq 10: quantum 1514b
 Sent 14926 bytes 193 pkts (dropped 0, overlimits 0)
qdisc prio 1: bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 401836 bytes 488 pkts (dropped 0, overlimits 0)
```

Todo o tráfego adicional foi para o 10:, que é a qdisc com maior prioridade. Nenhum tráfego foi enviado para qdisc de prioridades menores, o que previamente aconteceu com todo o scp.

*A qdisc CBQ*

Como dito anteriormente, CBQ é a qdisc mais complexa disponível, a mais exagerada, a menos compreendida, e provavelmente a mais difícil de se fazer funcionar. Isso não ocorre porque os autores são incompetentes, longe disso, é que o algoritmo CBQ não é tão preciso e não combina direito com a forma como o Linux trabalha.

Além de ser uma fila de classe, a qdisc CBQ também é um limitador de banda e é nesse aspecto que ele não funciona tão bem. Ela deveria funcionar assim: se tentarmos limitar a banda de uma conexão de 10 Mbit/s para 1 Mbit/s , o link deveria ficar 90% ocioso o tempo todo. Se não ficar, precisa-se regular para que ele fique ocioso 90% do tempo.

Isso é realmente difícil de medir, mas o CBQ na verdade deriva o tempo ocioso do número de microsegundos que passam entre requisições por mais dados que vem da camada de hardware. Combinados, isso pode ser usado para estimar quão cheio ou vazio o link está.

Isso é meio tortuoso e nem sempre chega a um resultado apropriado. Por exemplo, se a velocidade do link atual de uma interface não for capaz de transmitir numa velocidade real de 100 Mbit/s de dados, talvez por um driver mal implementado ou alguma outra limitação. Uma placa de rede PCMCIA quase nunca atinge a velocidade de 100 Mbit/s por causa do modo como o barramento é projetado.

Isso ainda fica pior quando se considera interfaces não reais como links PPP over Ethernet ou PPTP em cima de TCP/IP. A banda efetiva no caso, é provavelmente determinada pela eficiência de pipes no userspace – que são imensos.

*Limitação com CBQ em detalhes*

Como dito anteriormente, o CBQ trabalha tendo certeza de que o link está ocioso por tempo suficiente para diminuir a banda real para a taxa configurada. Para fazer isso, ele calcula o tempo médio que passa entre os pacotes.

Durante suas operações, o tempo ocioso efetivo é medido usando um EWMA (média exponencial de movimento por peso), que considera pacotes recentes como sendo exponencialmente mais importantes do que os que passaram. A média de carga do Unix é calculada da mesma forma.

O tempo ocioso calculado é subtraído do EWMA medido, e o número resultante é chamado de “avgidle”. Um link perfeitamente balanceado tem “avgidle” igual a zero: pacotes chegam exatamente a cada intervalo calculado.

Um link sobrecarregado tem “avgidle” negativo, e se for muito negativo, CBQ desliga por um tempo e está em “overlimit”.

Reciprocamente, um link ocioso pode aumentar muito o avgidle, o que permitiria banda infinita após algum tempo de silêncio entre as conversões. Para prevenir isso, o “avgidle” é limitado no “maxidle”.

Se em *overlimit*, na teoria, o CBQ poderia regular ele mesmo para o tempo exato que foi calculado entre a passagem de dois pacotes, e então passar um pacote, e depois regular-se novamente.
Parâmetros utilizados em CBQ:

Avgpkt

Tamanho médio do pacote, medido em bytes. É preciso desse campo para calcular o “maxidle”, que é derivado do “maxburst”, que é especificado nos pacotes.

Bandwidth

A banda física do dispositivo, necessária para se calcular o tempo ocioso.

Cell

O tempo que um pacote leva para ser transmitido através de um dispositivo pode aumentar em passos, baseado no tamanho do pacote. Um pacote entre 800 e 806 de tamanho pode levar o mesmo tempo para ser transmitido, por exemplo. Na maioria das vezes é definido como 8. Precisa ser um múltiplo de 2.

Maxburst

Esse número de pacotes é utilizado para calcular o valor “maxidle” para que quando o “avgidle” atingir o “maxidle”, esse número de pacotes médios possa ser processado antes que o “avgidle” volte a ser 0. Ele deve ser

definido como um número alto para ser mais tolerante ao tráfego de rajadas. Pode-se definir o “maxidle” diretamente, somente com esse parâmetro.

Minburst

Como mencionado anteriormente, o CBQ precisa se regular no caso de overlimit. A solução ideal para fazer isso para o tempo ocioso exato calculado, e passar 1 pacote. Para kernels Unix, contudo, geralmente é difícil agendar eventos em um tempo menor do que 10 ms, então é melhor regular para um período maior, e então passar os pacotes de uma vez só.

Minidle

Se o avgidle é menor que 0, isso caracteriza uma situação de overlimit e é preciso esperar até que o avgidle seja grande o suficiente para enviar um pacote. Para impedir que uma rajada repentina derrube o link por um período longo, o valor de avgidle é igualado ao minidle (se o primeiro diminuir muito).

Minidle é especificado em microsegundos negativos, então 10 significa que avgidle é regulado para – 10μs.

MPU

Minimum packet size – necessário pois mesmo um pacote de tamanho zero é enviado com 64 bytes em uma rede ethernet, e então leva um certo tempo para transmitir. O CBQ precisa saber isso para calcular de forma precisa o tempo ocioso do sistema.

Rate

Tamanho da taxa desejada de saída da qdisc – é conhecida como “speed knob”.

Internamente, CBQ tem muitos parâmetros de sintonia fina. Por exemplo, as classes que conhecidamente não tem dados enfileirados a elas não são consultadas. Classes em overlimit são penalizadas diminuindo sua prioridade efetiva.

Comportamento de classe do CBQ

Além de limitar, utilizando as já mencionadas aproximações de tempo ocioso, o CBQ também atua como uma fila PRIO no sentido de que cada classe tem diferentes prioridades e que uma prioridade menor será processada antes de uma prioridade maior.

Cada vez que um pacote é requisitado pela camada de hardware para ser enviado a rede, um processo de round-robin por peso começa, começando com um número de prioridade mais baixo da classe.

Eles são então agrupados e perguntados se eles têm dados disponíveis. Se têm, ele é retornado. Depois que é permitido a uma classe a desenfileirar um número de bytes, a próxima classe dentro daquela prioridade é tentada.

Os parâmetros seguintes controlam o processo WRR (Weighted Round Robin):

Allot

Quando a CBQ externa é requisitada parar enviar um pacote à interface, ela irá tentar todas as qdiscs internas (nas classes), na ordem do parâmetro “priority”. Cada vez que uma classe recebe a sua vez de enviar dados, ela só pode enviar uma certa quantidade de dados. “Allot” é a unidade base dessa quantidade de dados.

Prio

A fila CBQ pode também agirá como um periférico PRIO. Classes interiores com prioridade mais alta são tentadas primeiro, desde que tenham tráfego.

Weight

Weight ou Peso é o que ajuda o processo WRR. Cada classe recebe a chance de enviar. Se existem classes que significantemente tem mais banda que outras classes, faz sentido permitir que elas enviem mais dados em uma vez do que os outros.

O CBQ adiciona todos os pesos em uma classe e as normaliza. Então pode-se usar números arbitrários: só as relações são importantes. O peso renormalizado é multiplicado pelo parâmetro “allot” para determinar a quantidade de dados que pode ser enviado em um round.

Todas as classes numa hierarquia CBQ nunca compartilham o mesmo número principal.

Parâmetros CBQ que determinam o compartilhamento de link e empréstimo

Além de puramente limitar certos tipos de tráfego, ele também possibilita especificar quais classes podem emprestar a capacidade de outras classes, ou, em outras palavras, emprestar banda.

Isolado/Compartilhado

Uma classe que é configurada como isolada não irá emprestar banda a seus irmãos. Isso é usado quando se tem grupos competitivos ou não-amigáveis no link que não querem dar chances um ao outro.

O programa de controle “tc” também entende o significado de “sharing” ou compartilhar, como sendo o inverso do isolado.

Bounded/Borrow

Uma classe pode ser “bounded” , o que significa que ela não irá tentar emprestar banda de suas classes irmãs. O “tc” também conhece o borrow ou empréstimo que é o inverso do “bounded”.

Uma situação típica poderia ser quando se tem duas empresas usam um mesmo link e ambas são isoladas e “bounded”, o que significa que elas são realmente limitadas a sua taxa de banda delimitada, e também não irão emprestar banda umas as outras.

Numa situação dessas, podem haver outras classes que são permitidas a trocar banda.

Exemplo de Configuração:

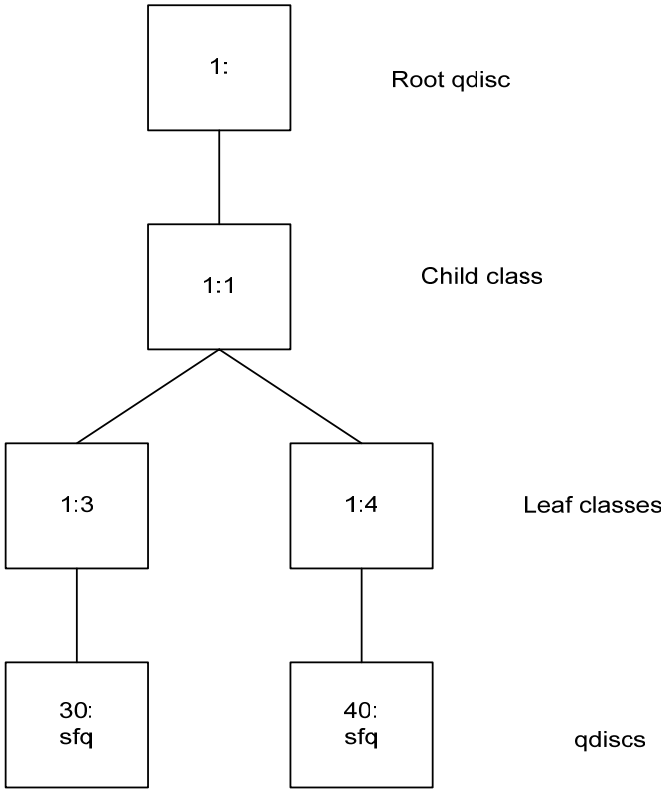


**Figura 2.5 - Exemplo de fila CBQ**

A configuração limita o tráfego de um servidor Web a 5 Mbit/s, e um servidor SMTP a 3 Mbit/s. Juntos, eles não podem passar de 6 Mbit/s. Uma interface de rede ethernet de 100 Mbps e as classes podem pegar banda emprestada uma de outra.

```
# tc qdisc add dev eth0 root handle 1:0 cbq bandwidth 100Mbit        \
  avpkt 1000 cell 8
# tc class add dev eth0 parent 1:0 classid 1:1 cbq bandwidth 100Mbit  \
  rate 6Mbit weight 0.6Mbit prio 8 allot 1514 cell 8 maxburst 20      \
  avpkt 1000 bounded
```

Essa parte instala a qdisc root e a classe 1:1 de costume. A classe 1:1 é “bounded”, então o total de banda não pode exceder 6 Mbit/s.

```
# tc class add dev eth0 parent 1:1 classid 1:3 cbq bandwidth 100Mbit  \
  rate 5Mbit weight 0.5Mbit prio 5 allot 1514 cell 8 maxburst 20      \
  avpkt 1000
# tc class add dev eth0 parent 1:1 classid 1:4 cbq bandwidth 100Mbit  \
  rate 3Mbit weight 0.3Mbit prio 5 allot 1514 cell 8 maxburst 20      \
  avpkt 1000
```

Essas são as duas folhas. O peso e a taxa configurada foram escalados. Ambas as classes não são bounded, mas são conectadas a classe 1:1, que é bounded. Então, a soma das bandas das duas classes não podem nunca passar de 6 Mbit/s.

```
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
# tc qdisc add dev eth0 parent 1:4 handle 40: sfq
```

Ambas as classes tem qdiscs FIFO por padrão. Mas serão trocadas por filas SFQ para que cada fluxo de dados seja tratado de forma igual.

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip \
  sport 80 0xffff flowid 1:3
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip \
  sport 25 0xffff flowid 1:4
```

Esses comandos, anexados diretamente a qdisc raiz, enviam os tráfegos para as qdiscs certas.

Se o tráfego não for classificado por nenhuma das duas regras, os dados serão processados pelo 1:0 e então serão ilimitados.

Se o SMTP+Web juntos tentarem exceder ao tráfego definido no limite de 6 Mbits/s, a banda será dividida de acordo com o parâmetro weight, dando 5/8 do tráfego para o servidor web e 3/8 do tráfego para o servidor de e-mail.

Com essa configuração, também pode-se dizer que o tráfego do servidor Web sempre terá um mínimo de 5/8 * 6 Mbit/s = 3.75 Mbit/s.

Outros parâmetros CBQ: split e defmap

Uma qdisc classfull precisa chamar os filtros para determinar a qual a qual classe um pacote será enfileirado.

Além de chamar o filtro, o CBQ também oferece outras opções: defmap & split. Como normalmente se quer filtrar o flag TOS somente, uma sintaxe especial é provida. Sempre que o CBQ precisar descobrir aonde os pacotes precisam ser enfileirados, ele checa se esse nó é um “split node” ou nó dividido. Se for, uma das sub-qdiscs terá indicada que deseja receber todo o tráfego contendo uma certa prioridade configurada, como derivado do campo TOS, ou opções de socket definidas por outras aplicações.

Os bits de prioridade do pacote são definidas com o campo defmap para ver qual correspondência existe. Em outras palavras, é uma forma mais rápida de se criar um filtro, que apenas vê se existe correspondência a certas prioridades. Um defmap de FF (hex) irá verificar tudo. Um mapa de 0, nada. Uma configuração simples pode ajudar a ver as coisas de forma mais clara.

```
# tc qdisc add dev eth1 root handle 1: cbq bandwidth 10Mbit allot 1514 \
  cell 8 avpkt 1000 mpu 64
# tc class add dev eth1 parent 1:0 classid 1:1 cbq bandwidth 10Mbit    \
  rate 10Mbit allot 1514 cell 8 weight 1Mbit prio 8 maxburst 20        \
  avpkt 1000
```

Defmap refere aos bits TC_PRIO, que são definidos assim:

| TC_PRIO. | NUM | Corresponds to TOS |
|---|---|---|
| BESTEFFORT | 0 | Maximize Reliability |
| FILLER | 1 | Minimize Cost |
| BULK | 2 | Maximize Throughput (0x8) |
| INTERACTIVE_BULK | 4 | |
| INTERACTIVE | 6 | Minimize Delay (0x10) |
| CONTROL | 7 | |

**Tabela 2.4 - Definição dos bits TC_PRIO**

O número TC_PRIO, corresponde aos bits, contados da direita.

Agora as classes interativas e bulk (tráfego de menor prioridade):

```
# tc class add dev eth1 parent 1:1 classid 1:2 cbq bandwidth 10Mbit     \
  rate 1Mbit allot 1514 cell 8 weight 100Kbit prio 3 maxburst 20        \
  avpkt 1000 split 1:0 defmap c0
# tc class add dev eth1 parent 1:1 classid 1:3 cbq bandwidth 10Mbit     \
  rate 8Mbit allot 1514 cell 8 weight 800Kbit prio 7 maxburst 20        \
  avpkt 1000 split 1:0 defmap 3f
```

O split-qdisc é 1:0, que é onde a escolha será feita. C0 é o binário para 11000000, 3F para 00111111, então os dois juntos irão corresponder a tudo. A primeira classe corresponde aos bits 7 & 6, e depois correspondem ao "interativo" e "tráfego de controle". A segunda classe corresponde ao resto.

O nó 1:0 agora tem uma tabela como essa:

| priority | send to |
|---|---|
| 0 | 01:03 |
| 1 | 01:03 |
| 2 | 01:03 |
| 3 | 01:03 |
| 4 | 01:03 |
| 5 | 01:03 |
| 6 | 01:02 |
| 7 | 01:02 |

**Tabela 2.5 - Prioridades do nó 1:0**

É possível também passar um "change mark", que indica exatamente quais prioridades podem ser trocadas. Só é necessário usar isso se o comando "tc class change" estiver sendo usado. Por exemplo, para um tráfego de melhor esforço para 1:2, poderia ser assim:

```
# tc class change dev eth1 classid 1:2 cbq defmap 01/01
```

O mapa de prioridades em 1:0 mostra isso:

| priority | send to |
|---|---|
| 0 | 01:02 |
| 1 | 01:03 |
| 2 | 01:03 |
| 3 | 01:03 |
| 4 | 01:03 |
| 5 | 01:03 |
| 6 | 01:02 |
| 7 | 01:02 |

**Tabela 2.6 – Prioridades do nó 1:0 após troca de prioridades**

*Hierarchical Token Bucket*

Martin Devera [Devera, 2002] percebeu que CBQ é complexo e não é otimizado para várias situações. Sua aproximação hierárquica se adequa bem a configurações, enquanto se tem uma taxa de banda fixa que se quer dividir para diferentes objetivos, dando a cada uma delas uma garantia de banda [Floyd, 1995], com a possibilidade de especificar a quantidade de banda que será emprestada.

O HTB funciona como CBQ, mas não utiliza do tempo ocioso para limitar a banda. Ao invés disso, ele é uma token bucket filter classful – apesar do nome. Tem apenas alguns poucos parâmetros, que são bem documentados.

Quando sua configuração HTB vai ficando mais complexa, a configuração fica bem escalada. Com o CBQ já é complexo até nos casos mais simples. O HTB é parte do kernel oficial 2.4.20 ou superiores. Contudo, talvez ainda seja preciso pegar um patch para adequar o HTB para o "tc". O HTB do kernel e o userspace têm que ser usados na mesma versão, ou o "tc" não funcionará com o HTB.

Exemplos de configuração

A funcionalidade é quase idêntica ao do CBQ:

```
# tc qdisc add dev eth0 root handle 1: htb default 30
# tc class add dev eth0 parent 1: classid 1:1 htb rate 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:10 htb rate 5mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:20 htb rate 3mbit ceil 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1kbit ceil 6mbit burst 15k
```

O autor (Martin Devera) recomenda o SFQ para paras as filas abaixo.

```
# tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
# tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
# tc qdisc add dev eth0 parent 1:30 handle 30: sfq perturb 10
```

Filtros que direcionarão o tráfego para as classes corretas tem que ser adicionados:

```
# U32="tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32"
# $U32 match ip dport 80 0xffff flowid 1:10
# $U32 match ip sport 25 0xffff flowid 1:20
```

O HTB certamente parece maravilhoso – se as classes 10: e 20:, ambas têm suas bandas garantidas, e existe mais para dividir, elas pegam emprestado a uma relação de 5:3, como era de se esperar.

O tráfego não classificado é roteado para o 30:, que tem pouca banda sozinho, mas que pode pegar emprestado se ela acabar. Como o SFQ foi escolhido internamente, tem-se um comportamento justo (round-robin).

2.2.1.6 Classificando os pacotes com filtros

Para determinar qual classe pode processar um pacote, uma “cadeia classificadora” ou “classifier-chain” é chamada cada vez que uma escolha precisa ser feita. Essa cadeia consiste de todos os filtros anexados a uma qdisc classful.

Na Figura 2.3, quando se enfileira um pacote, cada ramo do filtro é consultado por uma instrução relevante. Uma configuração típica seria ter um filtro 1:1 que direcione os pacotes a 12: e um filtro no 12: que direcione os pacotes para 12:2.

Pode-se também anexar essa regra depois para 1:1, mas pode-se ganhar eficiência tendo mais testes específicos abaixo da cadeia.

E mais uma vez – pacotes são apenas enfileirados de cima para baixo. Quando eles são desenfileirados, eles vão para a interface novamente. Eles não caem do fim da árvore para o adaptador de rede.

Alguns exemplos simples de filtros

Suponha que se tenha uma fila PRIO chamada “10:” que contém 3 classes, e que se deseja designar todo o tráfego de e para a porta 22 para a banda de maior prioridade. Os filtros ficariam dessa forma:

```
# tc filter add dev eth0 protocol ip parent 10: prio 1 u32 match \
  ip dport 22 0xffff flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 1 u32 match \
  ip sport 80 0xffff flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 2 flowid 10:2
```

Isso significa: anexe a eth0, nó 10: a prioridade 1 u32 um filtro que corresponda ao IP na porta de destino 22 e envie tudo para a banda 10:1. E depois ele repete o mesmo para tudo com origem na porta 80. O último comando diz que tudo que não tiver correspondência nos filtros até agora deve ir para a banda 10:2, a segunda banda de maior prioridade.

Selecionando o endereço IP:

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 \
  match ip dst 4.3.2.1/32 flowid 10:1
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 \
  match ip src 1.2.3.4/32 flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 2      \
  flowid 10:2
```

Isso designa o tráfego para 4.3.2.1 e o tráfego de 1.2.3.4 para a fila de prioridade máxima, e o resto para a de prioridade secundária.

Pode-se concatenar as correspondências, para corresponder ao tráfego de 1.2.3.4 e porta 80.

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 match ip src 4.3.2.1/32 \
  match ip sport 80 0xffff flowid 10:1
```

A maioria dos comandos de limitação começam com esse preâmbulo.

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 ..
```

Essas são as chamadas correspondências “u32”, que pode-se fazer correspondência com qualquer parte do pacote.

No endereço de origem/destino:

A máscara de origem corresponde ao IP de origem 1.2.3.0/24, “destination mask” corresponde ao IP 4.3.2.0/24. Para definir um único host, deve-se usar /32 na máscara.

Na porta de origem/destino, todos os protocolos IP:

Origem: “match ip sport 80 0xffff, destination “, “match ip dport 80 0xffff”

No protocolo ip (tcp, udp, icmp, gre, ipsec):

Devem-se usar os números do /etc/protocols. Por exemplo, ICMP é 1: “match ip protocol 1 0xff”

No fwmark

Pode-se marcar os pacotes com ipchains ou iptables e ter que marcar pacotes roteando entre as interfaces. Isso é realmente útil para o exemplo de limitação de banda na interface eth1 que veio da eth0:

```
# tc filter add dev eth1 protocol ip parent 1:0 prio 1 handle 6 fw flowid 1:1
```

Isso não é uma correspondência u32.

Pode-se marcar dessa forma:

```
# iptables -A PREROUTING -t mangle -i eth0 -j MARK --set-mark 6
```

O número 6 é arbitrário.

No campo TOS:

Para selecionar interatividade: minimum delay traffic:

```
# tc filter add dev ppp0 parent 1:0 protocol ip prio 10 u32 \
   match ip tos 0x10 0xff \
  flowid 1:4
```

Para tráfego não interativo, usa-se 0x08 0xff.

2.2.1.7 O dispositivo de enfileiramento intermediário (IMQ)

O dispositivo de enfileiramento intermediário não é um qdisc, mas sua utilização é firmemente voltado as qdiscs. Com o Linux, as qdiscs são adicionadas às interfaces de rede e tudo que é enfileirado a esses dispositivos é primeiramente enfileirado a qdisc. Nesse conceito, duas limitações acontecem:

1) Somente é possível limitar a banda de saída (existe qdisc ingress, mas as possibilidades são muito limitadas comparadas com qdiscs classful);
2) Uma qdisc pode apenas ver o tráfego em uma interface, não há como gerenciar o tráfego globalmente.

O IMQ: existe para ajudar a resolver essas duas limitações. Resumindo, pode-se por tudo quaisquer tipos de tráfego em uma qdisc. Especialmente pacotes marcados são interceptados pelos hooks do netfilter NF_IP_PRE_ROUTING e NF_IFPOST_ROUTING e passam por uma qdisc anexada ao dispositivo IMQ. Um alvo iptables é usado para marcar os pacotes.

Isso permite que se faça limitação do tráfego de entrada já que é possível marcar pacotes vindos de uma origem e/ou tratar as interfaces como classes para definir limites globais. Pode-se também fazer várias outras coisas como colocar o tráfego HTTP em uma qdisc, colocar novas requisições de conexões em uma qdisc, etc.

Exemplo de Configuração

A primeira coisa que deve ser feita é usar a limitação ingress para dar a si mesmo uma maior garantia de banda. A configuração é como qualquer outra interface.

```
tc qdisc add dev imq0 root handle 1: htb default 20
tc class add dev imq0 parent 1: classid 1:1 htb rate 2mbit burst 15k
tc class add dev imq0 parent 1:1 classid 1:10 htb rate 1mbit
tc class add dev imq0 parent 1:1 classid 1:20 htb rate 1mbit
tc qdisc add dev imq0 parent 1:10 handle 10: pfifo
tc qdisc add dev imq0 parent 1:20 handle 20: sfq
tc filter add dev imq0 parent 10:0 protocol ip prio 1 u32 match \
                ip dst 10.0.0.230/32 flowid 1:10
```

Nesse exemplo, u32 é usado para classificação. Outros classificadores também devem funcionar como esperado. Agora o tráfego tem que ser selecionado e marcado para ser enfileirado para a imq0.

```
ip link set imq0 up
```

O alvo do IMQ do iptables é válido para as cadeias PREROUTING e POSTROUTING da tabela mangle. A sintaxe é:

```
IMQ [ --todev n ] n : number of imq device
```

O tráfego não é enfileirado quando chega ao alvo, mas sim depois disso. A exata localização do tráfego que entra no dispositivo IMQ depende da direção do tráfego (entrando ou saindo). Esses são os hooks pré-definidos usados pelo iptables:

```
enum nf_ip_hook_priorities {
    NF_IP_PRI_FIRST = INT_MIN,
    NF_IP_PRI_CONNTRACK = -200,
    NF_IP_PRI_MANGLE = -150,
    NF_IP_PRI_NAT_DST = -100,
    NF_IP_PRI_FILTER = 0,
    NF_IP_PRI_NAT_SRC = 100,
    NF_IP_PRI_LAST = INT_MAX,
};
```

Para o tráfego de entrada, o IMQ registra a si mesmo com a prioridade NF_IP_PRI_MANGLE + 1 o que significa que os pacotes entram no dispositivo IMQ diretamente após passarem pela cadeia PREROUTING da tabela MANGLE do iptables.

Para o tráfego de saída, o IMQ usa o NF_IP_PRI_LAST que honra o fato dos pacotes descartados pela tabela de filtros não ocuparem banda.

### 2.2.2 QoS em OpenBSD

O enfileiramento no OpenBSD é chamado pelo firewall “pf” [OpenBSD Man, 2004]. O OpenBSD utiliza organizadores ou agendadores (schedulers) para fazer controle de banda. Por padrão, o OpenBSD usa um scheduler FIFO (First In First Out) [OpenBSD PF FAQ, 2005]. Uma fila FIFO funciona como um caixa de supermercado -- o primeiro item na fila é o primeiro a ser processado. Conforme novos pacotes chegam vão sendo colocados no fim da fila. Caso a fila fique cheia, e aqui a analogia com o supermercado termina, novos pacotes vão sendo descartados. Isso é conhecido como tail-drop [RFC2309, 1998].

O OpenBSD suporta dois agendadores:

- Filas Baseadas em Classe
- Fila de Prioridade

#### 2.2.2.1 Filas Baseadas em Classe

Filas Baseadas em Classe ou Class Based Queuing (CBQ) é um algoritmo de enfileiramento que divide a largura de banda entre múltiplas filas ou classes. Cada fila tem seu tráfego atribuído com base em endereço de origem ou destino, número de porta, protocolo, etc. Uma fila pode opcionalmente ser configurada para emprestar

banda de sua fila pai, caso ela não esteja utilizando sua largura total. As filas também recebem uma prioridade de forma que as que contêm tráfego interativo como SSH, podem ter seus pacotes processados antes de filas contendo tráfego normal, como FTP.

Filas CBQ são organizadas de forma hierárquica. No topo da hierarquia está a fila raiz que define a quantidade total de banda disponível. As outras filas são criadas sob a fila raiz e cada uma delas pode receber parte da largura de banda da fila raiz. Por exemplo, filas podem ser definidas assim:

Fila Raiz (2Mbps)
- Fila A (1Mbps)
- Fila B (500Kbps)
- Fila C (500Kbps)

Neste caso, a largura de banda total disponível está definida para 2 megabits por segundo (Mbps). Esta banda é então dividida entre outras três filas.

A hierarquia pode ser expandida definindo-se filas dentro de filas. Para dividir a banda igualmente entre diferentes usuários e ainda classificar o tráfego de forma que certos protocolos não ocupem a banda de outros, uma estrutura parecida com essa pode ser definida:

Fila Raiz (2Mbps)
- Usuário A (1Mbps)
  - ssh (50Kbps)
  - bulk (950Kbps)
- Usuário B (1Mbps)
  - audio (250Kbps)
  - bulk (750Kbps)
  - http (100Kbps)
  - outros (650Kbps)

A cada nível a soma do total de banda vinculada a cada fila nunca é maior que o valor da banda da fila raiz.

Uma fila pode ser configurada para emprestar banda de sua fila pai caso as outras filas dentro da fila pai não estejam usando sua porção e a banda esteja disponível, como na configuração a seguir:

Fila Raiz (2Mbps)
- Usuário A (1Mbps)
  - ssh (100Kbps)
  - ftp (900Kbps, borrow)
- Usuário B (1Mbps)

Se o tráfego na fila ftp exceder os 900Kbps e o tráfego na fila Usuário A for menor que 1 Mbps (porque a fila ssh está usando menos que 100 Kbps), a fila ftp irá emprestar banda de Usuário A. Desta forma a fila ftp pode usar mais banda do que foi definido a princípio, quando necessário. Quando o tráfego na fila ssh aumenta, a banda emprestada é devolvida.

O CBQ atribui a cada fila um nível de prioridade. Filas com prioridade alta têm preferência sobre outras com menor prioridade em caso de congestionamento, contanto que estejam contidas na mesma fila pai (em outras palavras, contanto que estejam no mesmo nível na hierarquia). Filas com a mesma prioridade são processadas em seqüência (round-robin). Por exemplo:

Fila Raiz (2Mbps)
    Usuário A (1Mbps, priority 1)
        ssh (100Kbps, priority 5)
        ftp (900Kbps, priority 3)
    Usuário B (1Mbps, priority 1)

O CBQ irá processar as filas Usuário A e Usuário B em seqüência - nenhuma fila será preferida sobre a outra. Enquanto a fila Usuário A está sendo processada, o CBQ também irá processar suas subfilas. Neste caso, a fila ssh tem prioridade maior e terá tratamento preferencial sobre a fila ftp caso a rede fique congestionada. As filas ssh e ftp não tem suas prioridades comparadas às filas Usuário A e Usuário B pelo fato de elas não estarem no mesmo nível na hierarquia.

2.2.2.2 Fila de Prioridade

Fila de Prioridade ou Priority Queuing (PRIQ) designa várias filas numa interface de rede com cada uma tendo um único nível de prioridade. Uma fila com alta prioridade é sempre processada na frente de uma fila com prioridade menor.

A estrutura de enfileiramento no PRIQ é simples – não se pode definir filas dentro de filas. A fila raiz é definida onde é configurado o total de banda disponível, então subfilas são definidas sob a raiz. Exemplo:

Fila raiz (2Mbps)
    Fila A (priority 1)
    Fila B (priority 2)
    Fila C (priority 3)

A fila raiz é definida possuindo 2 Mbps de largura de banda disponível e três subfilas são definidas. A fila com maior prioridade (o maior número na prioridade) é servida primeiro. Quando todos pacotes nesta fila são processados, ou caso a fila esteja vazia, o PRIQ vai para a próxima fila com prioridade mais alta. Dentro da fila, os pacotes são processados num sistema FIFO (First In First Out).

Quando se usa PRIQ deve-se planejar as filas com muito cuidado. Como PRIQ sempre processa a fila com prioridade mais alta primeiro, é possível que uma fila com alta prioridade faça com que pacotes destinados a outra fila com prioridade menor atrasem muito ou até mesmo sejam descartados caso a fila preferida esteja recebendo pacotes constantemente.

2.2.2.3 Random Early Detection

Random Early Detection (RED) é um algoritmo para evitar congestionamento. Seu trabalho é evitar congestionamento na rede certificando-se de que a fila nunca fique cheia. Ele realiza a tarefa calculando continuamente o tamanho médio da fila e comparando-a com dois valores, um valor mínimo e um valor máximo. Se o tamanho médio da fila estiver abaixo do valor mínimo então nenhum pacote será descartado. Se a média estiver

acima do valor máximo então todos os pacotes que chegarem serão descartados. Se a média estiver entre os dois valores então os pacotes são descartados baseados no cálculo da probabilidade obtido do tamanho médio da fila. Em outras palavras, conforme o tamanho médio da fila se aproxima do valor máximo, mais pacotes são descartados. Ao descartar pacotes, o RED escolhe aleatoriamente de quais conexões ele irá descartá-los. Conexões usando grandes porções da largura de banda têm maior probabilidade de terem seus pacotes descartados.

O RED é útil porque evita uma situação conhecida como sincronização global e pode acomodar aumentos repentinos no tráfego. Sincronização global refere-se a uma queda na capacidade de transferência devido aos pacotes descartados de várias conexões ao mesmo tempo. Por exemplo, caso o congestionamento ocorra num roteador transmitindo tráfego para 10 conexões FTP e os pacotes de todas (ou quase todas) as conexões são descartados (como é o caso com enfileiramento FIFO), a capacidade média de transferência cairá severamente. Esta não é uma situação ideal porque faz com que todas as conexões FTP reduzam sua taxa de transferência e também significa que a rede não é utilizada mais em todo seu potencial. RED evita esse cenário escolhendo aleatoriamente quais conexões terão pacotes descartados ao invés de escolher todas elas. Conexões usando muita largura de banda têm chances maiores de terem seus pacotes descartados. Desta forma, conexões que ocupam muita banda serão evitadas, o congestionamento será evitado e não ocorrerão sérias perdas nas taxas de transferência. Além disso, RED pode responder a aumentos repentinos no tráfego pelo fato de começar a descartar pacotes antes de a fila ficar cheia. Quando o tráfego inesperado chegar, existirá espaço suficiente na fila para armazenar os novos pacotes.

RED deve ser usado apenas quando o protocolo de transporte for capaz de responder a indicadores de congestionamento da rede. Na maioria dos casos isto significa que RED deve ser usado para enfileirar tráfego TCP e nunca tráfego UDP ou ICMP.

#### 2.2.2.4 Explicit Congestion Notification

Explicit Congestion Notification (ECN) [RFC3168, 2001] trabalha em conjunto com RED para notificar dois hosts comunicando-se pela rede de qualquer congestionamento no caminho de comunicação. Para fazer isso RED insere um marcador (flag) no cabeçalho do pacote ao invés de descartá-lo. Supondo que o host enviando dados tem suporte a ECN, ele pode identificar essa marcação e reduzir seu tráfego de acordo.

#### 2.2.2.5 Configurando Filas

Desde o OpenBSD 3.0 a implementação de filas Alternate Queueing (ALTQ) se tornou parte do sistema base. Desde o OpenBSD 3.3, o ALTQ foi integrado ao PF. A implementação ALTQ do OpenBSD suporta os organizadores Class Based Queueing (CBQ) e Priority Queueing (PRIQ). Ele também suporta Random Early Detection (RED) e Explicit Congestion Notification (ECN).

Pelo fato de ALTQ ter sido integrado ao PF, o PF deve estar habilitado para que o sistema funcione.

Filas são configuradas no arquivo pf.conf. Existem dois tipos de diretivas utilizadas para configuração de filas:

altq on - habilita enfileiramento na interface, define qual organizador (scheduler) usar e cria a fila raiz
queue - define as propriedades da subfila

A sintaxe para a diretiva altq on é:

```
altq on interface scheduler bandwidth bw qlimit qlim \
        tbrsize size queue { queue_list }
```

interface - a interface de rede onde o enfileiramento deve ser ativado;
scheduler - o organizador a ser utilizado. Valores possíveis são cbq e priq. Somente um organizador por vez pode estar ativo na interface;
bw - o valor total de banda disponível para o organizador. Isto pode ser especificado como um valor absoluto utilizando os sufixos b, Kb, Mb, e Gb para representar bits, kilobits, megabits, e gigabits por segundo, respectivamente ou como uma percentagem da banda na interface;
qlim - o número máximo de pacotes que podem ser armazenados na fila. Este parâmetro é opcional. O padrão é 50;
size - o tamanho do regulador token bucket em bytes. Caso não seja especificado, o tamanho é definido baseado na largura de banda da interface;
queue_list - uma lista de subfilas a serem criadas sob a fila raiz.

Por exemplo:

```
altq on fxp0 cbq bandwidth 2Mb queue { std, ssh, ftp }
```

Isso habilitará o CBQ na interface fxp0. A largura de banda total disponível é configurada para 2 Mbps. Três subfilas são definidas: std, ssh e ftp.

A sintaxe para a diretiva queue é:

```
queue name [on interface] bandwidth bw [priority pri] [qlimit qlim] \
        scheduler ( sched_options ) { queue_list }
```

name - o nome da fila. Este nome deve ser um dos nomes de filas definidos na diretiva altq on queue_list. Para o cbq também pode ser nome de fila definido numa diretiva queue_list numa queue anterior. Nomes de fila não devem ser mais longos que 15 caracteres;
interface - a interface de rede onde a fila é válida. Este valor é opcional, e quando não especificado, tornará a fila válida em todas interfaces;
bw - a quantidade total de banda disponível para a fila. Pode ser especificado como um valor absoluto ou usando sufixos b, Kb, Mb, e Gb representando bits, kilobits, megabits, e gigabits por segundo, respectivamente ou um valor em porcentagem da fila principal. Este parâmetro só é aplicável quando usar o organizador CBQ. Se não for especificado, o padrão é 100% da banda da fila pai;
pri - a prioridade da fila. Para o CBQ a faixa de prioridade varia de 0 a 7 e para PRIQ a faixa vai de 0 a 15. Prioridade 0 é a mais baixa. Quando não especificada, o valor padrão de 1 é usado;
qlim - o número máximo de pacotes que podem ser armazenados na fila. Quando não for especificado, o padrão 50 é usado;
scheduler - o organizador utilizado, CBQ ou PRIQ. Deve ser o mesmo da fila raiz.

sched_options - outras opções podem ser passadas para controlar o comportamento do organizador:

- default - define uma fila padrão onde os pacotes que não casarem com nenhuma outra fila serão armazenados. Uma fila padrão é necessária;
- red - habilita Random Early Detection (RED) na fila;
- rio - habilita RED com IN/OUT. Neste modo, RED manterá múltiplos valores médios da fila e múltiplos valores de comparação, um para cada nível de Qualidade de Serviço IP;
- ecn - habilita Explicit Congestion Notification (ECN) na fila. ECN implica RED;
- borrow - a fila pode emprestar banda de sua fila pai. Esta opção só pode ser utilizada com o organizador CBQ;
- queue_list - uma lista de subfilas a serem criadas dentro desta fila. Uma queue_list pode ser definida apenas quando o organizador utilizado for CBQ;

Continuando o exemplo acima:

```
queue std bandwidth 50% cbq(default)
queue ssh bandwidth 25% { ssh_login, ssh_bulk }
        queue ssh_login bandwidth 25% priority 4 cbq(ecn)
        queue ssh_bulk bandwidth 75% cbq(ecn)
queue ftp bandwidth 500Kb priority 3 cbq(borrow red)
```

Aqui os parâmetros das subfilas previamente definidas são configurados. A fila std possui largura de banda de 50% da fila raiz (ou 1 Mbps) e está configurada como a fila padrão. A fila ssh possui 25% da banda da fila raiz (500 Kb) e também contém duas filas ssh_login e ssh_bulk. A fila ssh_login tem prioridade maior que ssh_bulk e ambas têm ECN habilitado. A banda atribuída à fila ftp é de 500 Kbps com prioridade 3. Ela também pode tomar banda emprestada quando estiver disponível e também tem RED habilitado.

Cada definição de fila tem sua banda especificada. Sem especificar a banda, o PF dará à fila 100% da banda da fila pai. Nesta situação, que causará um erro quando as regras são carregadas uma vez que existe uma fila com 100% da banda, nenhuma outra fila pode ser definida neste nível uma vez que não existe banda livre para alocar.

2.2.2.6 Atribuindo Tráfego a uma Fila

Para atribuir tráfego a uma fila, a palavra-chave queue é usada em conjunto com as regras de filtragem do PF. Por exemplo, um conjunto de regras de filtragem contendo uma linha como:

```
pass out on fxp0 from any to any port 22
```

Pacotes que casarem com esta regra podem ser enviados para uma fila específica através do uso da palavra-chave queue:

```
pass out on fxp0 from any to any port 22 queue ssh
```

Ao usar a palavra queue com diretivas de bloqueio, quaisquer pacotes TCP RST ou ICMP Unreachable resultantes serão enviados à fila especificada.

As designações de filas podem ocorrer numa interface diferente da definida na diretiva altq on:

```
altq on fxp0 cbq bandwidth 2Mb queue { std, ftp }
queue std bandwidth 500Kb cbq(default)
queue ftp bandwidth 1.5Mb
pass in on dc0 from any to any port 21 queue ftp
```

Enfileiramento está habilitado em fxp0 mas a designação acontece em dc0. Se os pacotes combinando com a regra pass saírem pela interface fxp0, serão enfileirados na fila ftp. Este tipo de enfileiramento pode ser muito útil em roteadores.

Normalmente apenas um nome de fila é informado com a palavra queue, mas caso um segundo nome seja especificado a fila será usada para pacotes com um Type of Service (ToS) com pouco atraso e para pacotes TCP ACK sem payload de dados. Um bom exemplo disso pode ser obtido utilizando SSH. Sessões de login SSH irão definir o ToS para low-delay enquanto que sessões SCP e SFTP não. O PF pode usar estas informações para enfileirar pacotes pertencentes a uma conexão de login numa fila diferente dos pacotes que não forem de conexões de login. Isso pode ser útil para priorizar os pacotes das conexões de login sobre pacotes de transferência de arquivos.

```
pass out on fxp0 from any to any port 22 queue(ssh_bulk, ssh_login)
```

Isto envia pacotes pertencentes a conexões de login SSH para a fila ssh_login e pacotes pertencentes a conexões SCP e SFTP para a fila ssh_bulk. Conexões de login SSH terão seus pacotes processados antes dos pacotes SCP e SFTP, porque a fila ssh_login tem prioridade maior.

Atribuir pacotes TCP ACK a uma fila de alta prioridade é útil em conexões assimétricas, isto é, conexões que possuem taxas de upload e download diferentes, como conexões ADSL, por exemplo. Numa conexão ADSL, se o canal de upload estiver sendo utilizado em sua capacidade máxima e um download é iniciado, o download irá sofrer porque os pacotes TCP ACK que devem ser enviados entrarão num congestionamento quando tentarem passar pelo canal de upload. Testes têm mostrado que para atingir os melhores resultados, a largura de banda na fila de upload deve ser configurada para um valor menor do que a conexão realmente é capaz. Por exemplo, se uma conexão ADSL tem uma taxa máxima de upload de 640 Kbps, configurar a banda da fila raiz para um valor como 600 Kbps deve resultar numa melhora de performance.

Ao utilizar a palavra queue com regras que mantém o estado das conexões keep state como:

```
pass in on fxp0 proto tcp from any to any port 22 flags S/SA \
        keep state queue ssh
```

O PF gravará uma entrada para a fila na tabela de estado, de forma que pacotes saindo pela fxp0 que combinem com a conexão stateful serão enviados para a fila ssh. Perceba que apesar da palavra queue estar sendo utilizada num regra filtrando tráfego de entrada, o objetivo é especificar uma fila para o tráfego que sai; a regra acima não enfileira pacotes de entrada.

2.2.2.7 Exemplo de Configuração

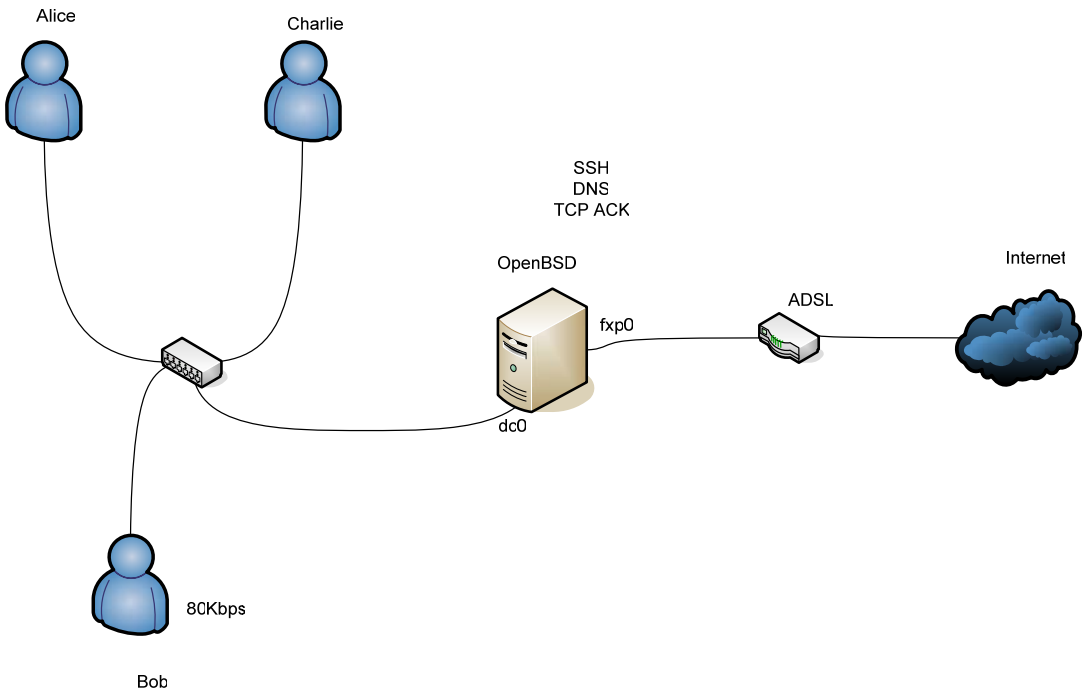


**Figura 2.6 - Exemplo de rede com controle de banda - OpenBSD**

Neste exemplo, o OpenBSD está sendo usado num gateway Internet para uma pequena rede doméstica com três estações de trabalho. O gateway faz filtragem de pacotes e NAT. A conexão Internet é via linha ADSL rodando a 2Mbps de downstream e 640Kbps de upstream.

A política de filas para esta rede:

1) Reservar 80 Kbps de largura de banda de download para Bob, para que ele possa jogar online sem sofrer lag (falta de interatividade por causa de congestionamentos na rede) por causa dos downloads de Alice ou Charlie. Permitir ao Bob usar mais de 80Kbps quando houver disponibilidade;
2) SSH interativo e mensagens instantâneas terão maior prioridade do que tráfego comum;
3) Consultas e respostas DNS terão a segunda maior prioridade;
4) Pacotes TCP ACK saindo terão maior prioridade que qualquer outro tráfego de saída.

Abaixo, segue o conjunto de regras adequado à política da rede, para o exemplo da Figura 2.6

```
# Habilita enfileiramento na interface externa para controlar tráfego
# indo para Internet. Usa o organizador PRIQ para controlar somente prioridades.
# Define largura de banda para 610Kbps para obter melhor performance
# na fila TCP ACK.

altq on fxp0 priq bandwidth 610Kb queue { std_out, ssh_im_out, dns_out, \
        tcp_ack_out }

# Define os parâmetros para as subfilas.
# std_out - a fila padrão. qualquer regra abaixo que não especifique
#           explicitamente uma fila terá seu tráfego inserido nesta fila.
# ssh_im_out - SSH interativo e tráfego de várias mensagens instantâneas.
# dns_out - pesquisas DNS.
# tcp_ack_out - pacotes TCP ACK sem dados de payload.
```

```
queue std_out     priq(default)
queue ssh_im_out  priority 4 priq(red)
queue dns_out     priority 5
queue tcp_ack_out priority 6

# Habilita enfileiramento na interface interna para controlar tráfego
# vindo da Internet. Usa organizador CBQ para controlar a banda.
# A largura de banda máxima é 2Mbps.

altq on dc0 cbq bandwidth 2Mb queue { std_in, ssh_im_in, dns_in, bob_in }

# Define os parâmetros para as subfilas.
# std_in      - a fila padrão. Qualquer regra de filtragem abaixo que
#               não especifique explicitamente uma fila terá seu
#               tráfego inserido nesta fila.
# ssh_im_in - SSH interativo e tráfego de várias mensagens instantâneas.
# dns_in - respostas DNS.
# bob_in - banda reservada para a estação de trabalho de Bob.
#              permitir a ele emprestar banda dos outros.

queue std_in    bandwidth 1.6Mb cbq(default)
queue ssh_im_in bandwidth 200Kb priority 4
queue dns_in    bandwidth 120Kb priority 5
queue bob_in    bandwidth 80Kb cbq(borrow)

# Na seção de filtragem do arquivo de configuração

alice        = "192.168.0.2"
bob          = "192.168.0.3"
charlie      = "192.168.0.4"
local_net    = "192.168.0.0/24"
ssh_ports    = "{ 22 2022 }"
im_ports     = "{ 1863 5190 5222 }"

# Regras de filtragem para entrada em fxp0
block in on fxp0 all

# Regras de filtragem para saída em fxp0
block out on fxp0 all
pass  out on fxp0 inet proto tcp from (fxp0) to any flags S/SA \
            keep state queue(std_out, tcp_ack_out)
pass  out on fxp0 inet proto { udp icmp } from (fxp0) to any keep state
pass  out on fxp0 inet proto { tcp udp } from (fxp0) to any port domain \
            keep state queue dns_out
pass  out on fxp0 inet proto tcp from (fxp0) to any port $ssh_ports \
            flags S/SA keep state queue(std_out, ssh_im_out)
pass  out on fxp0 inet proto tcp from (fxp0) to any port $im_ports \
            flags S/SA keep state queue(ssh_im_out, tcp_ack_out)

# Regras de filtragem de entrada em dc0
block in on dc0 all
pass  in on dc0 from $local_net

# Regras de filtragem de saída em dc0
block out on dc0 all
pass  out on dc0 from any to $local_net
pass  out on dc0 proto { tcp udp } from any port domain to $local_net \
            queue dns_in
pass  out on dc0 proto tcp from any port $ssh_ports to $local_net \
            queue(std_in, ssh_im_in)
pass  out on dc0 proto tcp from any port $im_ports to $local_net \
            queue ssh_im_in
pass  out on dc0 from any to $bob queue bob_in
```

### 2.2.3 QoS em FreeBSD

#### 2.2.3.1 Descrição

No FreeBSD, utiliza-se o dummynet [Rizzo, 2000] para se limitar o tamanho da banda. O Dummynet é uma ferramenta flexível que originalmente foi feita para testar protocolos de rede, e desde então, foi utilizado para controle de banda.

Ele simula/impõe as limitações de enfileiramento e banda, atrasos, pacotes perdidos e efeitos de caminhos múltiplos. Ele também implementa uma variante do "Weighted Fair Queueing", chamado WF2Q+. Ele pode ser usado em estações de trabalho ,ou em máquinas FreeBSD que funcionam como roteadores ou bridges.

Essas regras limitam o tráfego total ICMP para 50 Kbps:

```
ipfw add pipe 1 icmp from any to any
ipfw pipe 1 config bw 50Kbit/s queue 10
```

Essas regras limitam o tráfego de entrada para 300Kbits/s para cada host na rede 10.1.2.0/24:

```
ipfw add pipe 2 ip from any to 10.1.2.0/24
ipfw pipe 2 config bw 300Kbit/s queue 20 mask dst-ip 0x000000ff
```

Porém, para compartilhar para todas as máquinas um mesmo link, usa-se:

```
ipfw add queue 1 ip from any to 10.1.2.0/24
ipfw queue 1 config weight 5 pipe 2 mask dst-ip 0x000000ff
ipfw pipe 2 config bw 300Kbit/s
```

E essas regras simulam um link ADSL para um lugar distante (atraso de 1000 ms)

```
ipfw add pipe 3 ip from any to any out
ipfw add pipe 4 ip from any to any in
ipfw pipe 3 config bw 128Kbit/s queue 10 delay 1000ms
ipfw pipe 4 config bw 640Kbit/s queue 30 delay 1000ms
```

O dummynet trabalha interceptando pacotes (selecionados pelas regras do ipfw – ipfw é um dos firewalls do FreeBSD) em seus caminhos pela pilha de protocolos, e passando-os por um ou mais objetos denominados enfileiradores e pipes, os quais podem simular os efeitos das limitações de banda, propagação de atrasos, enfileiramentos de tamanho limitado, pacotes perdidos e caminhos múltiplos. Pipes são canais fixos de banda. Filas representam uma alternativa às filas de pacotes convencionais, associadas com um peso, as quais compartilham a banda do pipe a que estão conectadas, proporcionalmente ao seu peso.

Cada pipe e fila pode ser configurado separadamente, de modo que pode-se aplicar diferentes limitações/atrasos a diferentes tráfegos, de acordo com as regras do ipfw (ex: selecionando os protocolos, endereços e intervalo de portas, interfaces, etc.). Pipes e filas podem ser criadas dinamicamente. Usando-se apenas um único conjunto de regras pode-se aplicar limitações independentes para todos os hosts em uma subrede ou para todos os tipos de tráfego. Pode-se também configurar o sistema para construir cascatas de pipes, simulando, assim, redes com múltiplos links e caminhos entre a(s) origem(s) e o(s) destinatário(s).

2.2.3.2 Performance, status e disponibilidade

Diferentemente de outros limitadores de tráfegos de pacotes, os quais são executados no ambiente do usuário, o dummynet tem muito pouco overhead, tendo todo o seu processamento feito dentro do kernel.

O Dummynet tem sido amplamente utilizado por muitas pessoas, e o código é extremamente estável e robusto, especialmente a partir da versão 3.4-STABLE. Os consertos de bugs são geralmente aplicados a source tree do FreeBSD e estão disponíveis via CVS ou em versões do FreeBSD pré-compiladas.

2.2.3.3 Usando o dummynet (ipfw)

O dummynet é totalmente controlado pelos comandos do ipfw [Korff, 2005] e uma série de varáveis sysctl.

A estrutura básica dos comandos do ipfw é:

ipfw add [N] [prob X] action PROTO from SRC to DST [options]

Onde:

- N: é o número da regra;
- X: é um número entre 0 e 1 que, quando presente, indica a probabilidade de ter uma correspondência na regra se todos os campos estiverem corretos;
- Action: é uma das ações executadas ao haver uma correspondência, que pode ser permitir, negar, pular para outra regra, fila N (pipe N) entre outras. Para enviar um pacote a um pipe dummynet é preciso usar o pipe N;
- PROTO: é o tipo de protocolo a qual a regra irá se referir (IP, TCP, UDP, etc);
- SRC e DST: são os endereços de origem e destino (pode-se utilizar endereços com máscaras de rede e seguido por portas ou intervalo de portas);
- Options: podem ser usadas para definir o comportamento do tráfego vindo ou indo a alguma interface específica ou utilizando algum flag TCP, ou opções ICMP.

2.2.3.4 Variáveis Sysctl

O firewall é praticamente controlado pelo ipfw, e as variáveis do sysctl só servem para configuração global e parâmetros padrões. As principais variáveis sysctl [Lucas, M., 2003] para controlar o comportamento do ipfw, bridging e dummynet encontram-se a seguir:

net.inet.ip.fw.enable: 1

Habilita o firewall na pilha IP.

net.inet.ip.fw.one_pass: 1

Força uma única passagem pelo firewall. Se definido em 0, os pacotes que saem de um pipe serão reinjetados no firewall começando com a regra subseqüente.

net.inet.ip.fw.dyn_buckets: 256 (somente leitura)

Tamanho da tabela de hash atual, usado para regras dinâmicas.

net.inet.ip.fw.curr_dyn_buckets: 256

Tamanho da tabela de hash desejada, usada para regras dinâmicas.

net.inet.ip.fw.dyn_count: 3

Número atual de regras dinâmicas (somente leitura).

net.inet.ip.fw.dyn_max: 1000

Número máximo de regras dinâmicas. Se esse limite é excedido, deve-se esperar que a regra expire antes de se criar uma nova.

net.inet.ip.fw.dyn_ack_lifetime: 300
net.inet.ip.fw.dyn_syn_lifetime: 20
net.inet.ip.fw.dyn_fin_lifetime: 20
net.inet.ip.fw.dyn_rst_lifetime: 5
net.inet.ip.fw.dyn_short_lifetime: 5

Tempo de vida (em segundos) para vários tipos de regras dinâmicas.

net.inet.ip.dummynet.hash_size: 64

Tamanho da tabela de hash para pipes dinâmicos.

net.inet.ip.dummynet.expire: 1

Apaga pipes dinâmicos quando eles se tornam vazios.

net.inet.ip.dummynet.max_chain_len: 16

Taxa máxima entre o número de filas dinâmicas e hash buckets.

Quando o número de filas em pipe se excede, pacotes que não correspondem a nenhuma regra serão todos colocados no mesmo pipe padrão.

Controlando o bridging

As bridges são exclusivamente controladas pelas variáveis sysctl

net.link.ether.bridge_cfg: ed2:1,rl0:1,

Define as interfaces na qual a bridge é habilitada, e os clusters aos quais elas pertencem.

net.link.ether.bridge: 0

Habilita a bridge.

net.link.ether.bridge_ipfw: 0

Habilita o ipfw para a bridge.

Configuração dos pipes e das filas

Esse comando é usado para criar ou reconfigurar um pipe. NN é o identificador numérico (entre 1 e 65535) do pipe.

ipfw [-s field] pipe [NN] show

Esse comando mostra os parâmetros de um pipe. Se o pipe é dinâmico, então todos os pipes dinâmicos criados por ele são listados. A lista pode ser bastante longa. A opção –s permite agrupar a lista em um dos quatro contadores associados ao pipe.

Ipfw pipe NN delete

Destrói um único pipe. Pacotes enviados para um pipe inexistente são descartados.

Ipfw pipe flush

Destrói todos os pipes.

Os parâmetros seguintes podem ser configurados para um pipe, adicionando o comando na linha de configuração do pipe.

Bandwidth: bw NNunit

NN: é a banda designada ao pipe, unidade (que deve seguir o número sem espaços no meio) que pode ser em bit/s, Kbit/s Mbit/s, Byte/s, Kbyte/s, Mbytes/s ou abreviações não ambíguas.

Uma banda de 0 (ou nenhuma banda) resulta em nenhuma limitação (ou seja, nenhuma fila será criada).

Queue size: queue NN [unit]

Define o tamanho da fila, em slots se somente NN for especificado, ou então em bytes ou kbytes. Quando não há mais espaço na fila, os pacotes são descartados. O tamanho padrão da fila é de 50 slots. A combinação de banda e o tamanho da fila influenciam o atraso da mesma. Quando se usa bandas muito baixas, não se usa filas muito grandes, pois isso pode gerar muitos segundos de atraso na fila.

Ao se fazer testes na interface de loopback deve-se tomar cuidado ao se especificar o tamanho da fila em pacotes para não gerar atrasos muito grandes.

Delay: delay NN ms

Define a propagação do atraso na fila, em milisegundos. Cada componente de atraso da fila é independente da propagação do atraso. Todos os atrasos são aproximadamente com uma granularidade de 1/HZ segundos (HZ é tipicamente 100).

Random Packet Loss: plr X

X é um número real entre 0 e 1 que causa o descarte aleatório de pacotes. Isso é feito normalmente para simular perdas nos links. O padrão é 0, ou seja, nenhuma perda.

Dynamic queue creation: mask ...

É possível associar a máscara ao pipe para que as limitações de banda e fila sejam forçadas separadamente para pacotes pertencendo a fluxos de dados diferentes.

O comando mask permite que se especifiquem quais partes de quais campos contribuem para identificar um fluxo:

[proto N] [src-ip N] [dst-ip N] [src-port N] [dst-port N]

Onde N é o bitmask onde os bits significativos são definidos como 1. Pode-se especificar uma ou mais máscaras. O parâmetro “all” significa que todos os campos são igualmente significativos.

O padrão (quando nenhuma máscara é especificada) é ignorar todos os campos, para que todos os pacotes sejam considerados como pertencentes ao mesmo fluxo.

Quando um fluxo novo é encontrado, uma nova fila (com a banda e o tamanho da fila especificados) é criada.

O número de filas dinâmicas que podem ser criadas dessa forma podem se tornar muito largas. Elas são acessadas por uma tabela hash, cujo tamanho pode ser definido usando os especificados NN dos buckets após o comando mask.

Para usar WF2Q+, pacotes têm que ser passados para filas que por sua vez devem estar conectadas a pipes. Os comandos ipfw abaixo controlam os pipes dummynet:

ipfw queue NN config ...

Esse comando é usado para criar ou reconfigurar uma fila. NN é o número do identificador (entre 1 e 65535) da fila.

ipfw queue NN delete

Destrói uma fila simples: todos os pacotes enviados a uma fila não existente são descartados.

ipfw queue flush

Destrói todas as filas.

Os parâmetros seguintes podem ser configurados para uma fila, adicionando-se o comando na linha de configuração da fila.

Pipe: pipe NN

NN é o identificador do pipe usado para regular tráfego.

Weight: weight NN

NN é o peso (1 a 100, padrão 1) associado a fila.

Per-Flow queueing: mask ...

A sintaxe é a mesma que para os pipes. Contudo, todas as filas criadas dinamicamente irão compartilhar as bandas dos pipes pais, de acordo com o peso.

## 2.3 QoS em Produtos Comerciais

### 2.3.1 PacketShaper

O PacketShaper é uma solução para controle de tráfego inline, composta de 2 módulos. O primeiro atua como um monitor, verificando e identificando o tráfego do segmento, medindo tempo de resposta, a utilização da banda e notificando o usuário em condições pré-definidas. O segundo módulo efetua a alocação de banda baseado na prioridade organizacional, aplicando assim o conceito de qualidade de serviço (QoS).

Os produtos para priorização de tráfego normalmente se baseiam em protocolos de código aberto. O Packetshaper possui mecanismo de priorização por enfileiramento próprio, entretanto ele entende CoS (Classe de Serviço) e ToS (Tipo de Serviço) ambos incorporados ao protocolo IP. O Packetshaper também entende Diffserv e todo o esquema de identificadores do MPLS, funcionando como um tradutor universal entre protocolos de priorização, detectando as intenções de um protocolo e perpetuando essas intenções em um protocolo diferente enquanto repassa o pacote.

Exemplo de gráfico de utilização – tráfego de saída.

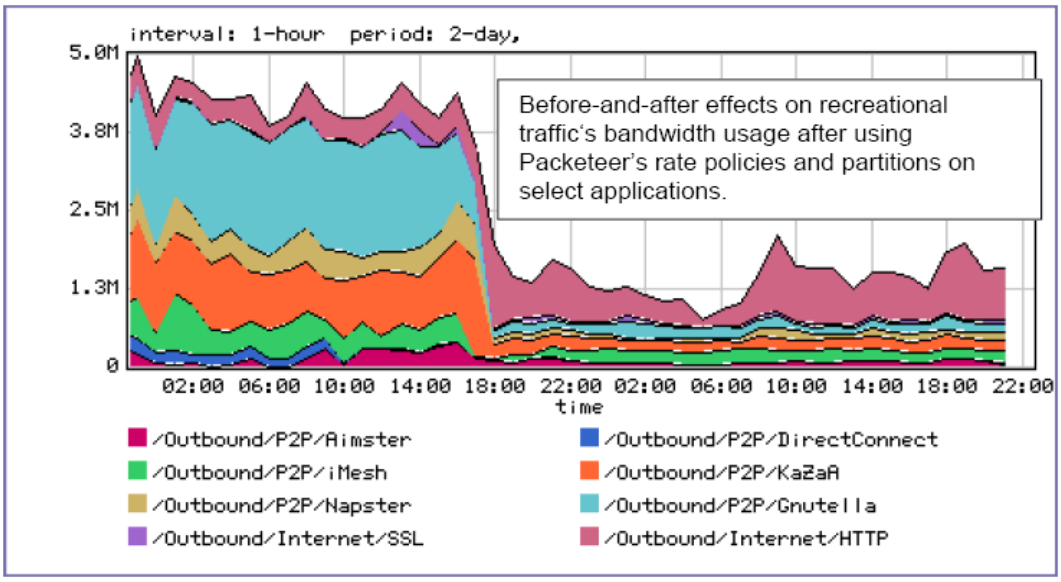


**Figura 2.7 – PacketShaper [Packeteer, 2005]**

No dia a dia, por ser um produto com foco exclusivo no controle de banda é considerado por muitos um produto extremamente eficiente e simples para o administrador de rede. Possui interface web e relatórios gráficos.

### 2.3.2 Sysmaster

O Sysmaster é um módulo colocado junto a um roteador que regula o tráfego de entrada e saída, utilizando diferentes formas de filtros e parâmetros, ou políticas restritivas para refinar a utilização de banda. O módulo de QoS facilita a entrega de pacotes priorizados ao servidor apropriado. O custo de um módulo desses ultrapassa o valor de trinta mil dólares.

O módulo de QoS inclui a funcionalidade de alocar banda dinamicamente a aplicações críticas. Utilizando de mecanismos de filtro e de enfileiramento, o tráfego de saída pode ser classificado e agendado usando diferentes políticas de tráfego, baseada nos mecanismos de DiffServ.

O módulo inclui agendadores avançados que priorizam o tráfego de saída baseada em sua prioridade pré-definida. O SysMaster pode também regular o tráfego de entrada aplicando filtros com políticas para refinar a utilização da banda.

Resumidamente, o SySMaster trabalha em camada 4 e pode utilizar-se de Diffserv.

### 2.3.3 Cisco

A Cisco é mundialmente conhecida pelos seus produtos na área de telecomunicações. É impossível ignorar a sua presença quando se fala de tratamento de tráfego, já que aproximadamente 80% do tráfego de dados do mundo passa por roteadores da Cisco [Diógenes, 2002].

Dessa forma, a Cisco possui inúmeros produtos. E grande parte deles, agrega alguma funcionalidade para prover algum tipo de qualidade de serviço.

Dentre essas funcionalidades, pode-se citar os algoritmos de WFQ (Weighted Fair Queueing) utilizado para alocar banda de forma seqüencial (round-robin fashion), o RED (Random Early Detection), que é um algoritmo para detectar possíveis congestionamentos que possam vir a ocorrer no link, e o CB-WFQ, que, para esse trabalho, talvez seja o mais importante a ser citado.

O CB-WFQ [Cisco, 2001] (Class-based Weighted Fair Queueing) é um algoritmo de controle de filas de classe utilizado pela Cisco para classificar o tráfego em diferentes classes da mesma forma que o CBQ o faz.

Dessa forma, a Cisco também implementa a maioria das tecnologias de enfileiradores, com a diferença de que esse não é o foco da maioria dos seus produtos.

Existem então, alguns poucos produtos comerciais que tratam exclusivamente de controle de banda, enquanto que na maioria das vezes, existem outros produtos (firewalls, roteadores etc) que têm algumas funcionalidades de controle de banda.

Já no software livre, existem algumas formas de se fazer. Esse projeto utilizará as características do CBQ para sua implementação, pois com esse algoritmo, além de se poder fazer subdivisões de classes na banda, também é possível trabalhar com garantia de banda. Dessa forma, o projeto mostrará, no capítulo seguinte, como será implementado o controle de banda usando CBQ, tanto no Linux como no OpenBSD.

# 3. Descrição do Projeto de gateway de borda proposto

Como parte do objetivo deste projeto é apresentada uma solução aberta para controle de banda em gateways de borda. A solução apresentada utiliza-se de software livre e dois sistemas operacionais de código-aberto: OpenBSD e Linux. Ambos serão utilizados e configurados para trabalharem como gateways de rede, fazendo limitações e controle de banda em camada 4. Na elaboração do projeto, foram adotados os seguintes passos:

1) Pesquisa de produtos comerciais e estudo de suas soluções para controle de banda, conforme descrito no capítulo 2;
2) Pesquisa de soluções abertas de controle de banda e viabilidade de implementação, descrita em parte no capítulo 2, e parte nesse capítulo;
3) Definição da topologia a ser implementada [RFC1918, 1996];
4) Definição e aquisição de hardware a ser utilizado [OpenBSD/i386, 2005];
5) Instalação dos sistemas operacionais com configurações voltadas para um gateway de internet;
6) Criação de scripts e regras para controle de banda;
7) Simulação e testes de download de arquivos, para verificação do correto funcionamento, descritos no capítulo 4, seções 4.5 à 4.7;
8) Comparação de consumo de hardware para as duas soluções apresentadas no próximo capítulo.

Esse estudo poderá fornecer uma base para que o público tenha acesso a soluções abertas de controle de banda, adequando apenas o modelo utilizado ao perfil de rede desejado.

Dessa forma, serão apresentadas duas soluções para limitação de banda, pois serão feitas verificações e testes para ver as vantagens e desvantagens de cada solução, tanto julgando o próprio desempenho da máquina ao tratar pacotes, como a facilidade de implementação da solução.

Em cada uma delas, o código-fonte do kernel será configurado, de forma a eliminar quaisquer funções, parâmetros ou infra-estruturas que não sejam utilizadas pela solução, deixando o kernel otimizado para a função que a solução exercerá.

## 3.1 Gateway Linux – Descrição

O que aqui denomina-se de primeira solução de controle de banda, é uma solução baseada em Linux, utilizando-se a distribuição Slackware Linux 10.1. A escolha da distribuição se baseia no fato de ser uma das distribuições que menos altera os códigos-fontes dos aplicativos que disponibiliza, mantendo assim a autenticidade do código-fonte.

Para o script de controle de banda criado, foram utilizados o shell BASH e o aplicativo “tc”, do pacote de aplicativos para roteamento avançado “iproute2”. Para que o “tc” possa gerenciar as filas, é necessária a recompilação do kernel [Welsh, 1997], habilitando as opções de roteamento avançado e de políticas para enfileiradores, além da definição de quais enfileiradores serão utilizados.

Além disso, para que a solução desempenhe seu papel com todos os recursos computacionais disponíveis, o kernel do sistema operacional Linux foi otimizado para atender os requisitos específicos deste projeto, de forma a retirar todas as opções que não fossem de uso exclusivo de um gateway de rede; ou que não tivessem relação com as políticas de controle de banda. Na tabela 3.1, mostra-se algumas das opções que foram desabilitadas no kernel do Linux.

| Opções do Kernel - Linux | |
|---|---|
| Suporte a dispositivos ainda em desenvolvimento | O suporte a dispositivos e drivers ainda em desenvolvimento foi desabilitado. |
| Utilização de Módulos | A utilização de módulos para carregar e descarregar partes do kernel após o sistema já ter sido iniciado é uma função extremamente útil em sistemas normais, mas para a solução criada, essa opção foi desabilitada. |
| Suporte a Notebooks/Laptops | O kernel do Linux possui alguns suportes especiais a Notebooks e Laptops de algumas empresas como Toshiba e Dell. Essas opções foram desabilitadas. |
| Emulador matemático | O emulador matemático em software foi desabilitado. |
| Suporte a Multiprocessamento | O suporte a multiprocessamento simétrico (SMP) foi desabilitado. |
| Porta Paralela | O suporte a porta paralela foi desabilitado. |
| Dispositivos Plug & Play | O suporte a dispositivos Plug & Play foi desabilitado |
| Rádio Amador | O suporte a dispositivos de Rádio Amador foram desabilitados |
| BlueTooth | O suporte a infraestrutura e dispositivos BlueTooth foram desabilitados |
| Kernel Hacking | A opção Kernel Hacking, no kernel do Linux, é a opção equivalente a Kernel Debugging no OpenBSD, e serve para gerar mensagens de depuração do código-fonte do kernel. Essa opção também foi desabilitada. |
| Processador Athlon/Duron/Sempron | Todo o sistema foi recompilado com instruções próprias a família Athlon/Duron/Sempron, da AMD. |
| Dispositivos de Controle de Energia | Todo o suporte a dispositivos de controle de energia, como APM ou ACPI foram desabilitados. |

**Tabela 3.1– Opções do Kernel do Linux Desabilitadas**

O arquivo de configuração do kernel com o todas as opções que foram desabilitadas detalhadas estão no Anexo 1 deste trabalho.

Após a otimização do kernel, e da correta instalação dos pacotes de aplicativos mencionados, foram estudadas as linguagens de script “bash shell scripting” e “korn shell scripting” [Michael, 2003], e criado um script com regras específicas limitando a banda na saída de uma das interfaces da solução.

## 3.2 Gateway OpenBSD – Descrição

A segunda solução de controle de banda utilizada baseia-se no sistema operacional OpenBSD. A escolha do sistema operacional se baseia no fato do TCP/IP ter tido suas origens nos BSD Unix, tendo dessa forma sua história de desenvolvimento diretamente ligada aos sistema operacionais baseados no padrão BSD 4.4. A escolha específica da vertente do OpenBSD deve-se ao fato deste ser mundialmente conhecido como o sistema operacional de código-aberto mais seguro existente atualmente.

Para utilização de controle de banda no OpenBSD será utilizado o framework ALTQ. O ALTQ é um framework que foi criado para as plataformas dos BSD Unix para controle de prioridade e alternância de filas, como o próprio nome já diz (Alternate Queue).

O OpenBSD desde a sua versão 3.3 fundiu o framework ALTQ no seu firewall PF. Desta forma, é preciso conhecer a sintaxe correta utilizada no firewall PF para se fazer o controle de banda necessário no OpenBSD.

Para ser possível a utilização do framework ALTQ no OpenBSD é necessário habilitá-lo no kernel do sistema. Desta forma, o código-fonte do kernel do sistema foi utilizado e totalmente recompilado, assim como a 1ª. solução apresentada. Dessa forma, retirou-se também do OpenBSD, todos os parâmetros, opções ou recursos que

não estivessem diretamente ligados a função de gateway ou de controle de filas. As opções que foram retiradas, encontram-se listadas na tabela abaixo.

| Opções do Kernel - OpenBSD | |
|---|---|
| Dispositivos de Controle de Energia | Suporte a dispositivos de controle de energia, como APM e ACPI foram desabilitados. |
| Processador i686 | O sistema foi todo recompilado utilizando instruções para processadores compatível com o padrão Intel 686 (que é o caso do Sempron) |
| Suporte a USB | Todo e qualquer suporte ao barramento USB 1.1 e 2.0 foi desabilitado. |
| Suporte a PCMCIA | Todo e qualquer suporte ao barramento PCMCIA foi desabilitado. |
| Suporte a Firewire | Todo e qualquer suporte ao barramento Firewire foi desabilitado. |
| Suporte a dispositivos SCSI | Tanto o suporte ao barramento SCSI, quanto a qualquer dispositivo SCSI I, II ou III foram desabilitados e retirados do kernel. |
| Suporte a RAID | Como não foi utilizado, todo suporte a "array" de discos foi desabilitado. |
| Placas de Rede Wireless | Todo e qualquer suporte a placas de rede Wireless foi desabilitado. |
| Dispositivos de Som | Todo o suporte do sistema para a utilização de som, assim como para os dispositivos (placas de som) de áudio foram desabilitados. |
| Protocolo IPV6 | O protocolo IP na sua versão 6 foi desabilitado. |
| Dispositivos ISDN | Suporte a dispositivos ISDN foi desabilitado. |
| Opções Criptográficas | Todas as opções de criptografia existentes no kernel foram desabilitadas. |
| Outros protocolos de Rede | Outros protocolos de rede que não o TCP/IP, como IPX ou Appletalk foram desabilitados. |
| Kernel Debugging | As opções de "debug" (depuração de código-fonte) de kernel foram desabilitadas. |
| Sistemas de Arquivos | Todo o suporte a sistemas de arquivos que não o FFS (4.4BSD), como FAT, NTFS, XFS, NFS, etc, foram desabilitados. |

**Tabela 3.2 - Opções do Kernel do OpenBSD desabilitadas**

O arquivo de configuração do kernel com o todas as opções que foram desabilitadas detalhadas estão no Anexo 2.

Após a customização do kernel, para este projeto foi estudada a sintaxe utilizada pelo firewall PF e pelo framework ALTQ, de forma a criar um conjunto de regras para o controle de banda de saída de uma das interfaces.

## 3.3 Definição do hardware utilizado no projeto

Como visto nas seções anteriores, o projeto apresenta dois tipos de solução que serão testadas, implementadas e comparadas. Então, tomou-se o cuidado de adotar hardwares compatíveis, visando a comparação das duas soluções ao fim do projeto. Com isto fez-se necessário encontrar dois hardwares idênticos. O não cumprimento dessa meta implicaria em influências nas medições de recursos das máquinas.

Dessa forma, foram adquiridos dois lotes de peças e foram montadas duas máquinas com a seguinte configuração cada uma: Gabinete ATX com fonte de 400W, Placa-mãe PCCHIPS M863G com Chipset SiS 741 de Northbridge e SiS 964L de Southbridge, Microprocessador AMD Sempron 2200, 256 Mb de memória RAM DDR com 333 MHz de clock, 1 unidade de disquete comum, 1 unidade de CD-ROM comum , 1 disco rígido de 30 Gb e 1

placa de rede sobressalente com o chipset VIA Rhine, pois a placa-mãe já possui uma interface de rede SiS 900 onboard..

Considerando que, para os fins propostos, a máquina definida tenha um poder de processamento maior do que o necessário para efetuar as tarefas que lhe serão delegadas [OpenBSD Archives, 2000], mas sem perder o foco no baixo custo, o hardware é novo e possui um bom desempenho para o projeto.

## 3.4 Infraestrutura física e topologia do projeto

A infraestrutura física montada é mostrada na figura abaixo:

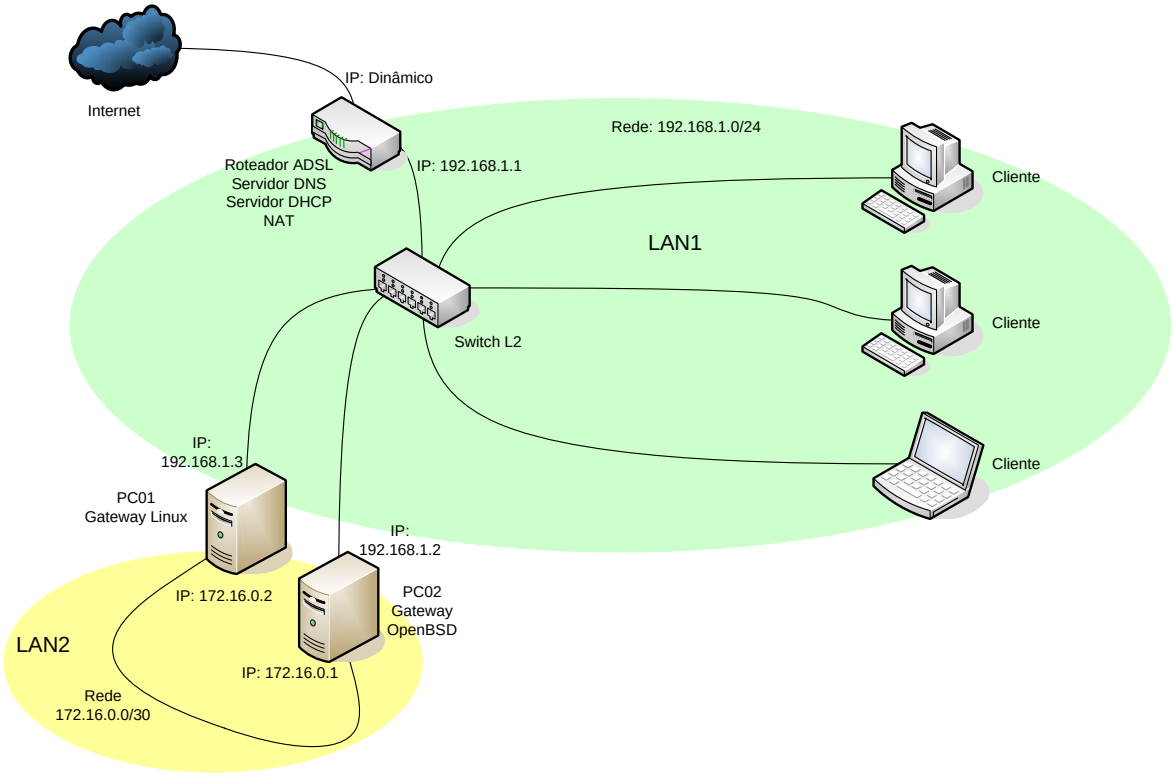


**Figura 3.1 – Diagrama Físico da Rede**

Foi utilizado para o projeto uma máquina com OpenBSD e outra com Slackware Linux. Porém, por uma questão de redundância e, para evitar quaisquer transtornos decorrentes de problemas físicos com as peças, os dois sistemas operacionais foram instalados nas duas máquinas, de forma que a solução funcionasse caso algo de errado acontecesse com uma das duas.

Dessa forma, definiu-se como PC-01 a máquina que foi utilizada como a solução 1 (Slackware Linux) e como PC-02 a máquina que foi utilizada como a solução 2 (OpenBSD).

A topologia definida para a solução prevê duas sub-redes para que o tráfego possa ser roteado de uma rede para outra pelo gateway, fazendo assim o controle de banda. A primeira rede é chamada de LAN1 é a rede 192.168.1.0/24. A segunda rede, chamada de LAN2, é a rede 172.16.0.0/30.

As duas soluções pertencem às duas redes. A primeira máquina, PC01 tem duas interfaces de rede e os IPs 192.168.1.2 em uma interface e 172.16.0.1 na outra. A segunda máquina, PC02, também tem duas interfaces de rede, e os IPs 192.168.1.3 em uma interface e 172.16.0.2 na outra.

O motivo de ter se designado máscara 255.255.255.252 para a segunda rede, é que a única comunicação que ocorrerá nela será entre as máquinas PC01 e PC02.

Na rede LAN1, existe um modem ADSL com IP externo recebido dinamicamente da prestadora de serviços de internet local e IP da interface interna definido como 192.168.1.1. O modem também provê um serviço de DNS (somente cache e redirecionamento) e um serviço DHCP, além de ter internamente um firewall e fazer NAT do seu endereço público para que as máquinas da rede interna possam ter acesso à internet. É nessa mesma rede que se encontram os clientes dos gateways.

A partir dessa topologia de rede, a implementação das duas soluções foi feita, conforme a Figura 3.1.

No próximo capítulo serão detalhadas as implementações das duas soluções aqui apresentadas.

# 4. Implementação do projeto de gateway

Conforme já foi dito, a implementação inicial dos dois sistemas tinha em mente utilizar uma máquina com um sistema operacional e a outra máquina com o outro, montando dois sistemas separados e independentes. Como forma de redundância apenas, os dois sistemas foram instalados nas duas máquinas. Porém, para a descrição do resto do projeto, tratar-se-á exclusivamente como PC01 a máquina com Linux, ou solução 1, e como PC02 a máquina com OpenBSD, ou solução 2.

## 4.1 Primeira solução – Slackware Linux

O Slackware 10.1 foi instalado nas duas máquinas. Para a instalação do Slackware Linux, utilizou-se o kernel **bare.i** para boot da instalação pelo CD-ROM e utilizou-se como mapa de teclado o tipo br-abnt2.map (ABNT 2). No Linux, não existe nenhum tipo de subparticionamento como no Unix baseados em BSD. As partições são criadas diretamente na tabela de partições.

Os discos rígidos IDE no Linux são identificados como “HD”, ou seja, como arquivo de dispositivo, ele ficaria como /dev/hd. No caso do disco rígido como Primary Master, o arquivo que identifica o dispositivo fica como /dev/hda. Pois a contagem, vai das letras A à D (em um ambiente IDE como, somente com um barramento primário e um secundário).

As partições seguem o seqüenciamento numérico após esse dispositivos. Ou seja, para o Linux, a partição criada do OpenBSD é identificada como /dev/hda1, por ser a primeira partição do disco rígido.

Dessa forma, foram criadas mais duas partições, ambas primárias. A primeira, /dev/hda2 foi criada com 300 Mb de espaço, e o tipo de partição foi definida como tipo 82, ou seja, “Linux Swap”. O tipo 82 é um tipo de partição padronizada para o Linux utilizar como espaço de memória virtual. A segunda partição criada, /dev/hda3, é onde o sistema operacional realmente será instalado, criada com 4 Gb de espaço, e o seu ponto de montagem é o / (root ou raiz). O esquema de particionamento utilizado foi o recomendado pelo manual do sistema operacional. O device /dev/hda1 não foi utilizado, pois nesse dispositivo está a partição de backup do OpenBSD.

A partição /dev/hda3 não foi definida como inicializável, ou seja, o flag de “bootable” não foi ativado. Isso não foi feito, porque durante a instalação do Slackware será feita a instalação do LILO, um gerenciador de inicialização, e o mesmo será instalado no MBR (Master Boot Record) do disco rígido. Dessa forma, essa definição de inicializável ou não, torna-se desnecessária quando se tem um menu interativo que pergunta qual partição se deve inicializar.

A partição /dev/hda3 foi formatada com o sistema de arquivos ReiserFS. Nenhum tipo de seleção fina de pacotes foi feita na instalação, de forma que foram instalados da categoria BASE, todos os aplicativos. Dessa forma, uma customização maior de pacotes será feita após o sistema estar instalado..

A instalação do LILO (gerenciador de inicialização) foi feita de forma automática durante a instalação, customizando-se apenas que a mesma seria feita no MBR (Master Boot Record), e que seria instalada de forma STANDARD, ou seja, sem a utilização de frame buffer (já que a mesma será desabilitada no kernel do sistema para o projeto). O script rc.pcmcia foi desabilitado já na instalação, já que não se utilizará quaisquer dispositivos que utilizem barramento PCMCIA no projeto.

A data e hora do sistema foram configuradas de acordo com o horário local, utilizando para isso a configuração do sistema para GMT -3h00.

## 4.2 Slackware Linux - Configuração

Algumas configurações foram feitas para adequar o sistema operacional à solução. Como só se instalou a parte básica do sistema, alguns pacotes foram instalados após a instalação básica:

iptables-1.2.11-i486-1.tgz
iproute2-2.6.9_ss040831-i486-1.tgz
openssh-3.9p1-i486-1.tgz
openssl-0.9.7e-i486-3.tgz
tcpip-0.17-i486-31.tgz

O pacote “tcpip-0.17-i486-31.tgz” é a suíte de aplicativos para o funcionamento do protocolo TCP/IP no userland Linux. Não basta apenas o suporte no kernel, é necessário também essa suíte de aplicativos para as configurações a serem feitas, como os endereços IP das placas de rede, gateways, rotas, DNS, etc. Os pacotes “iptables-1.2.11-i486-1.tgz” e “iproute2-2.6.9_ss040831-i486-1.tgz” foram instalados para que se possa utilizar as funções de controle de banda no Linux. Para enviar tráfegos para fila, é necessário o aplicativo “tc” que se encontra na suíte de aplicativos “iproute2-2.6.9_ss040831-i486-1.tgz”. O “openssl-0.9.7e-i486-3.tgz” foi instalado para que o “openssh-3.9p1-i486-1.tgz” possa ser utilizado, ou seja, para que exista no sistema um servidor SSH para acesso remoto. Dessa forma, alguns arquivos de inicialização do sistema também foram alterados para que o sistema já iniciasse com a configuração correta de rede. A seguir são apresentados os arquivos de configuração utilizados no projeto:

/etc/rc.d/rc.inet1.conf

```
# Config information for eth0:
IPADDR[0]="172.16.0.1"
NETMASK[0]="255.255.255.252"
USE_DHCP[0]=""
DHCP_HOSTNAME[0]=""
# Config information for eth1:
IPADDR[1]="192.168.1.2"
NETMASK[1]="255.255.255.0"
USE_DHCP[1]=""
DHCP_HOSTNAME[1]=""
GATEWAY="192.168.1.1"
```

Essas configurações se referem aos endereços IP das placas de rede (que no Linux são identificadas como eth0 e eth1) e ao gateway padrão. Além disso, existe também a configuração de DNS:

/etc/resolv.conf

```
search hadden.com.br
nameserver 192.168.1.1
```

Após isso, foi criado um arquivo /etc/rc.cbq com o script de configuração criado para o Linux:

/etc/rc.d/rc.cbq

```
# Apagando regras anteriores na device eth0
tc qdisc del dev eth0 root

# Cria qdisc principal (root qdisc)
tc qdisc add dev eth0 root handle 1:0 cbq bandwidth 100Mbit avpkt 1000 cell 8
```

```
# Cria class principal
tc class add dev eth0 parent 1:0 classid 1:1 cbq bandwidth 100Mbit \
      rate 100Mbit prio 8 allot 1514 cell 8 maxburst 20 avpkt 1000 bounded

# Cria classe 3
tc class add dev eth0 parent 1:1 classid 1:3 cbq bandwidth 100Mbit \
      rate 40Kbit prio 5 allot 1514 cell 8 maxburst 20 avpkt 1000 bounded

# Cria classe 4
tc class add dev eth0 parent 1:1 classid 1:4 cbq bandwidth 100Mbit \
      rate 400Kbit prio 5 allot 1514 cell 8 maxburst 20 avpkt 1000 bounded

# Cria classe 5
tc class add dev eth0 parent 1:1 classid 1:5 cbq bandwidth 100Mbit \
      rate 4Mbit prio 5 allot 1514 cell 8 maxburst 20 avpkt 1000 bounded

# Cria classe 6
tc class add dev eth0 parent 1:1 classid 1:6 cbq bandwidth 100Mbit \
      rate 40Mbit prio 5 allot 1514 cell 8 maxburst 20 avpkt 1000 bounded

# Troca fila da classe 3 para SFQ
tc qdisc add dev eth0 parent 1:3 handle 30: sfq

# Troca fila da classe 4 para SFQ
tc qdisc add dev eth0 parent 1:4 handle 40: sfq

# Troca fila da classe 5 para SFQ
tc qdisc add dev eth0 parent 1:5 handle 50: sfq

# Troca fila da classe 6 para SFQ
tc qdisc add dev eth0 parent 1:6 handle 60: sfq

# Cria filtro: tudo que sair pela porta 80 vai para a classe 3
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip sport 80 0xffff flowid 1:3

# Cria filtro: tudo que sair pela porta 81 vai para a classe 4
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip sport 81 0xffff flowid 1:4

# Cria filtro: tudo que sair pela porta 82 vai para a classe 5
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip sport 82 0xffff flowid 1:5

# Cria filtro: tudo que sair pela porta 83 vai para a classe 6
tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip sport 83 0xffff flowid 1:6
```

Após a criação desse arquivo, deu-se permissão “-rwxr-xr-x” para ele, ou seja, permissão para que ele possa ser executado. Editou-se então o arquivo /etc/rc.local, para que ele chamasse o arquivo /etc/rc.d/rc.cbq na inicialização do sistema.

Outro arquivo alterado foi o arquivo /etc/ssh/sshd_config, que é o arquivo que tem as configurações para o servidor SSH. Como foi utilizado o acesso remoto para testes e configurações da máquina, a configuração desse serviço precisa ser melhorada. Dessa forma, as opções abaixo foram habilitadas nesse arquivo:

```
PermitRootLogin no
Protocol 2
```

Essas configurações impedem que o usuário “root” faça logon remoto no sistema, de forma que para fazer administração remota é preciso logar-se como um usuário comum, que tenha permissão de se tornar “root” através do comando “su”.

O segundo parâmetro define que o serviço de SSH somente aceitará conexões SSH utilizando a versão 2 do protocolo SSH. Conexões com a versão 1 não serão aceitas.

Por causa dessa configuração, é necessário também criar um usuário para se logar ao sistema. O usuário criado foi o usuário “fabio”, e também foi criado um grupo chamado “hadden” para se criar de forma organizada quaisquer eventuais outros usuários que precisassem ser criados. Dessa forma, criou-se o grupo “hadden” com o group ID de número 2000. O usuário “fabio” foi criado com user ID 2001, e colocado no grupo “hadden” como grupo padrão, e no grupo adicional “wheel”.

Estar no grupo “wheel” é necessário a qualquer usuário comum para que ele possa trocar para o usuário “root” através do comando “su”.

Com essa configuração, espera-se conseguir um bom nível de segurança para se fazer acesso remoto, além de permitir as tarefas de roteamento entre as duas redes com o controle de banda.

## 4.3 Segunda Solução - OpenBSD

O OpenBSD 3.6 foi instalado, utilizando-se na instalação o tipo de terminal VT220, configuração de teclado PC-AT e tabela de codificação de teclado “BR” (ABNT 2). O OpenBSD foi a primeira partição criada em ambas as máquinas, ambas com 4 Gb de espaço e com o flag do tipo de partição no código hexadecimal A6 (identificação do sistema OpenBSD). As partições também foram configuradas como inicializáveis (flag de “boot” ativas).

Configuradas as partições, nos sistemas baseados no BSD Unix, também é preciso configurar os seus slices. Slices podem ser entendidos como um subparticionamento. Dessa forma, a partição que foi definida acima ficará subdividida em espaços de tamanhos diferentes, e são esses espaços que o sistema operacional utilizará como pontos de montagem.

Os slices utilizados foram os seguintes:

| Slice | Tamanho | Ponto de montagem |
|---|---|---|
| A | 150 Mb | / |
| B | 300 Mb | Swap |
| C | - | - |
| D | 200 Mb | /tmp |
| E | 200 Mb | /var |
| F | 500 Mb | /home |
| G | 1 Gb | /usr |

**Tabela 4.1 – Particionamento do OpenBSD**

O dispositivo do disco rígido no OpenBSD é identificado como “wd”. Como nas máquinas utilizadas, o disco rígido encontra-se como Master do barramento primário da controladora IDE, mais conhecido como Primary Máster e é o primeiro na contagem de dispositivos do kernel [OpenBSD FAQ, 2005]. Ou seja, no diretório de dispositivos, o arquivo de identificação do disco rígido no sistema atual é “wd0”.

Dessa forma, tem-se também os slices acima definidos como arquivos no diretório de devices:

Disco Rígido: /dev/wd0

- Slice A: /dev/wd0a
- Slice B: /dev/wd0b
- Slice C: /dev/wd0c
- Slice D: /dev/wd0d
- Slice E: /dev/wd0e
- Slice F: /dev/wd0f
- Slice G: /dev/wd0g

Tem-se então que o Slice A é o "/", ou seja, o root ou raiz do sistema. Por padrão, os kernels do OpenBSD são colocados na raiz do sistema, de forma que a primeira partição a ser lida, para que o sistema possa subir é o /. Dessa forma, fisicamente, o Slice A corresponde aos primeiros bytes do disco rígido.

O Slice B, no OpenBSD, é utilizado como Swap Space, ou mais conhecido como memória virtual. O Slice C não pode ser utilizado, pois no OpenBSD ele representa a identificação da partição inteira. O Slice D, definido como /tmp é o diretório onde o sistema operacional cria, apaga e se utiliza do espaço para fazer operações que necessitem de arquivos temporários. O Slice E, ou /var, é um espaço definido para ser utilizado com arquivos que variam, ou seja, spools de servidores de e-mail, arquivos de cache, logs do sistema, etc. O Slice F é o /home, onde ficam os arquivos de todos os usuários (exceto os do administrador do sistema). Finalmente, o Slice G é o /usr, onde ficam os arquivos de sistema, e que na atual topologia, foi definido como o maior dos slices porque para o projeto será necessária a recompilação do kernel do sistema, e é esse diretório que será utilizado tanto para espaço para o código-fonte descompactado do sistema, como para a compilação e criação do novo kernel. Todos os slices (exceto B e C, por serem de sistema) foram formatados com o tipo de sistema de arquivos 4.2BSD.

Os pacotes utilizados na instalação foram:

Bsd
Base36.tgz
Etc36.tgz
Misc36.tgz
Comp36.tgz
Man36.tgz

Os pacotes eliminados foram:

Bsd.rd
Bsd.mp
Game36.tgz
Xbase36.tgz
Xetc36.tgz
Xshare36.tgz
Xfont36.tgz
Xserv36.tgz

Bsd é o kernel padrão do sistema. É necessário que se instale algum kernel inicialmente. Base36.tgz é o pacote que contém as funcionalidades básicas do sistema, e é um pacote necessário na instalação. O etc36.tgz é um pacote que instala somente os arquivos de configuração do sistema. Porém, é também obrigatório na instalação. Misc36.tgz é um pacote que contém utilitários comuns na administração do sistema. Este foi instalado para utilização na implementação do projeto, porém, após o gateway estar pronto, ele é desnecessário. O mesmo acontece com os pacotes comp36.tgz e man36.tgz. O primeiro fornece todos os compiladores e bibliotecas necessárias para a compilação do sistema e do kernel, e o segundo fornece as páginas de manual online do sistema para todos os comandos e utilização geral do mesmo. Ambos são necessários para a implementação do projeto, porém, após o projeto pronto, ele não são necessários na tarefa exclusiva de tratamento de tráfego.

Os pacotes que não foram instalados, não farão qualquer diferença no sistema (além de ocupar espaço) nem durante a implementação do projeto, nem após isso. O pacote Bsd.mp não foi utilizado, pois é apenas o kernel com suporte a multiprocessamento. Como o processador utilizado é apenas um (AMD Sempron 2200), e além disso não possui função HT (hyper-threading) não é necessário utilizar-se deste kernel. O pacote Bsd.rd é um pacote que serve como emergência ou como utilização para debug de desenvolvedores do sistema, onde ao se iniciar o sistema com esse kernel, ele monta o sistema de arquivos raiz na memória RAM e não no disco. Game36.tgz é um pacote que contém apenas jogos. XBase36.tgz, Xetc36.tgz, Xfont36.tgz, Xserv36.tgz e Xshare36.tgz são pacotes para utilização do gerenciador de janelas X Window System. Como a função principal do sistema é tratar tráfego, não é necessária nenhuma interface gráfica de alta resolução para gerenciar os scripts.

Os sistema também foi configurado no fuso horário GMT -3h00, e o SSH foi habilitado de forma a permitir administração remota da máquina de forma segura.

## 4.4 OpenBSD - Configuração

Algumas configurações adicionais foram feitas no sistema. A configuração das placas de rede, no OpenBSD, foi feita utilizando os arquivos de configuração por placa pelos arquivos hostname.(nome da placa). No caso do OpenBSD:

/etc/hostname.sis0

```
# Placa de Rede SIS9000
192.168.1.3 255.255.255.0 NONE
```

/etc/hosname.vr0

```
# Placa de Rede Via Rhine
inet 172.16.0.2 255.255.255.252 NONE
```

O arquivo /etc/mygate, define o gateway padrão do sistema. Dessa forma, tem-se:

/etc/mygate

```
192.168.1.1
```

Para que as resoluções de DNS possam ser possíveis, a configuração do servidor DNS (no caso, o roteador ADSL) também foram definidas:

/etc/resolv.conf

```
lookup file bind
nameserver 192.168.1.1
```

Além disso, a configuração do nome da máquina também tem que ser feita:

/etc/myname

```
pc02.hadden.com.br
```

A configuração do arquivo /etc/sysctl.conf foi adequada à solução:

/etc/sysctl.conf

```
net.inet.ip.forwarding=1
```

Essa configuração serve para permitir o roteamento de pacotes entre as duas interfaces.

Os valores do arquivo /etc/gettytab foram alterados de:

```
P|Pc|Pc console:\
	:np:sp#9600:
```

Para:

```
P|Pc|Pc console:\
	:np:sp#9600:\
		:cl=\E[H\E[2J:
```

Essa configuração foi alterada para que toda vez que um usuário fizer logoff em uma console do sistema, a tela seja limpa e volte à tela de login, não deixando resquícios do que foi feito na tela.

A configuração dos serviços de SSH foram as mesmas do Linux:

/etc/ssh/sshd_config

```
PermitRootLogin no
Protocol 2
```

Para que o usuário “root” não possa fazer logon remoto e que só use o protocolo 2, conforme já explicado.

Dessa forma, é necessário também fazer a configuração de usuários e grupos, conforme no Linux. O mesmo foi feito, com o mesmo nome de usuário, e mesmos números de group ID e user ID do OpenBSD: fabio, uid=2001, hadden, gid=2000. A real diferença de ambiente entre os dois é que no OpenBSD utilizou-se por padrão o Korn Shell (ksh), para não se criar a necessidade de se instalar mais um pacote. No Slackware Linux, por padrão, o shell é o “bash” (Bourne Again Shell).

Além disso, o arquivo /etc/rc.conf foi editado, colocando-se a opção PF=YES, para que o firewall (PF) fosse carregado junto com a inicialização do sistema, carregando assim as regras de controle de banda juntamente com ele. Alterou-se, também, o padrão pf_rules=/etc/pf.conf para pf_rules=/etc/cbq.conf [OpenBSD Man, 2002],

para que o arquivo a ser lido na inicialização pelo firewall seja o /etc/cbq.conf, que é o arquivo onde se encontram as regras de controle de banda:

/etc/cbq.conf

```
# Define que a interface sis0 utilizará a técnica de enfileiramento
# CBQ em uma banda máxima de 100 megabits
altq on sis0 cbq bandwidth 100Mb queue { def, q1, q2, q3, q4 }

# Define a fila padrão do sistema com o tamanho de 100Kb
queue def cbq(default) bandwidth 100Kb

# Define a fila q1 com 40Kb de banda
queue q1 bandwidth 40Kb

# Define a fila q1 com 10Kb de banda
queue q2 bandwidth 400Kb

# Define a fila q1 com 20Kb de banda
queue q3 bandwidth 4Mb

# Define a fila q1 com 40Kb de banda
queue q4 bandwidth 40Mb

# Define que todo o tráfego que sair na interface sis0
# na porta 80, será enfileirado na fila q1
pass out on sis0 inet proto tcp from any port 80 to any keep state queue q1

# Define que todo o tráfego que sair na interface sis0
# na porta 81, será enfileirado na fila q2
pass out on sis0 inet proto tcp from any port 81 to any keep state queue q2

# Define que todo o tráfego que sair na interface sis0
# na porta 82, será enfileirado na fila q3
pass out on sis0 inet proto tcp from any port 82 to any keep state queue q3

# Define que todo o tráfego que sair na interface sis0
# na porta 83, será enfileirado na fila q4
pass out on sis0 inet proto tcp from any port 83 to any keep state queue q4
```

Além disso, também foi extraído para o diretório /usr/src/sys, o kernel do sistema, que foi recompilado.

## 4.5 Diagrama de Testes

Para fazer os testes do sistema, o seguinte ambiente foi criado:

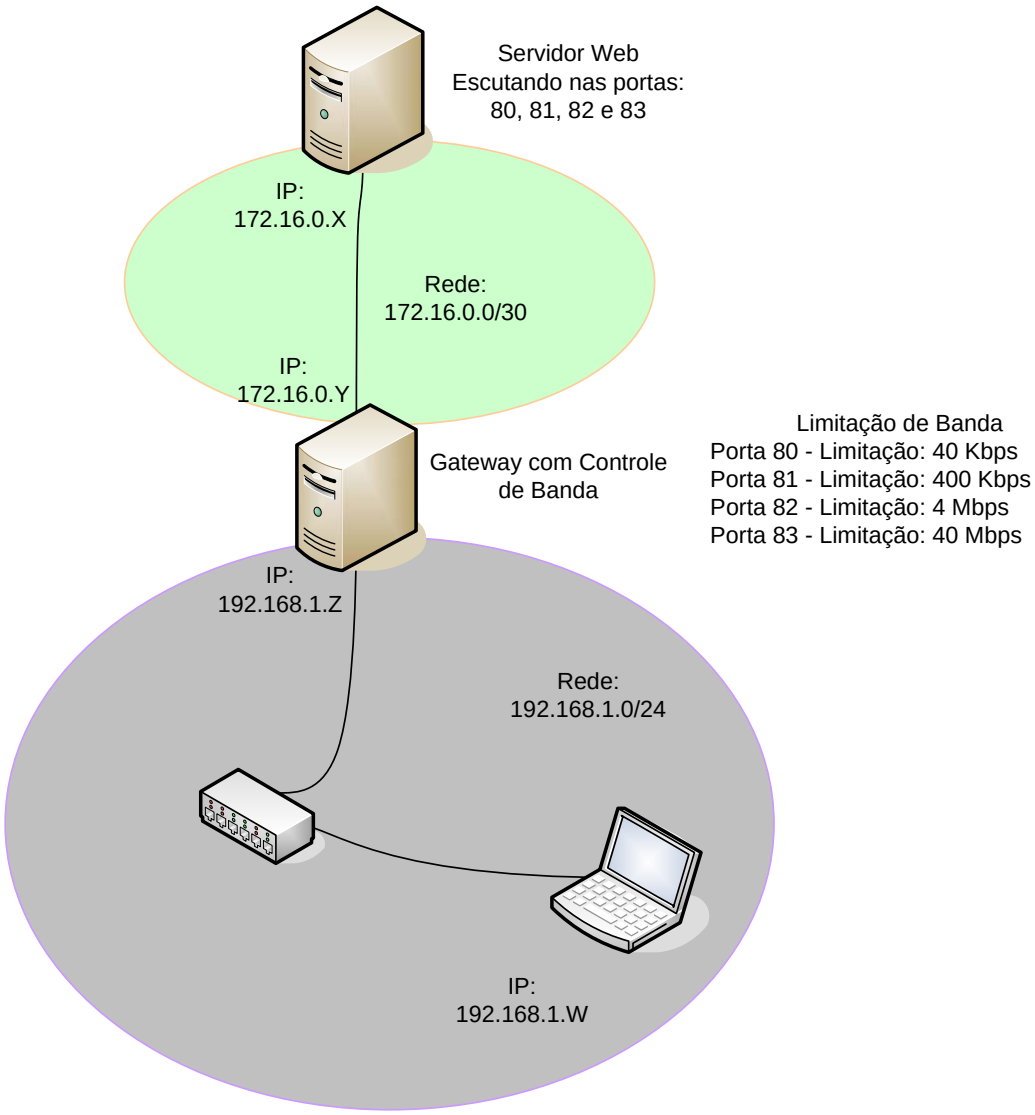


**Figura 4.1 – Diagrama de Testes**

Dessa forma, foi criado um servidor Web, utilizando-se para isso o servidor Apache. As letra X, Y, Z e W utilizadas na Figura 4.1, são as letras respectivas para cada ambiente, como será mostrado nas próximas sessões (Figuras 4.2 e Figura 4.8). A única configuração alterada do servidor foi a seguinte:

/etc/apache/httpd.conf

```
Listen IP:80
Listen IP:81
Listen IP:82
Listen IP:83
```

Essa configuração faz com que o servidor Web escute nas portas 80, 81, 82 e 83. Dessa forma, foi colocado um arquivo grande (maior do que 500 megabytes) chamado de teste.tar no diretório padrão onde esse servidor Web serve. Criou-se um arquivo “index.html” com os seguintes parâmetros:

```
<title>Pagina de Teste</title>
<html>
<body>
Links<br>
<a href=http://IP:80/teste.tar>40Kbps</a><br>
<a href=http://IP:81/teste.tar>400Kbps</a><br>
<a href=http://IP:82/teste.tar>4Mbps</a><br>
<a href=http://IP:83/teste.tar>40Mbps</a><br>
</body>
</html>
```

Dessa forma, o resultado da página é uma página com 4 links: um recebendo o arquivo teste.tar na porta 80, outro na porta 81, outro na porta 82 e outro na porta 83 do mesmo servidor.

No gateway, a banda da porta 80 estará sendo limitada a 40Kbps, a da porta 81 a 400 Kbps, a da porta 82 a 4Mbps e a da porta 83 a 40Mbps.

Com uma máquina de cliente, iniciou-se o download simultâneo dos 4 arquivos, analisando-se como cada solução comporta-se com o kernel padrão e com o kernel customizado.

## 4.6 Slackware Linux – Testes e Resultados

A Figura 4.2 abaixo mostra a topologia em que os testes da solução baseada no Slackware Linux foram feitos:

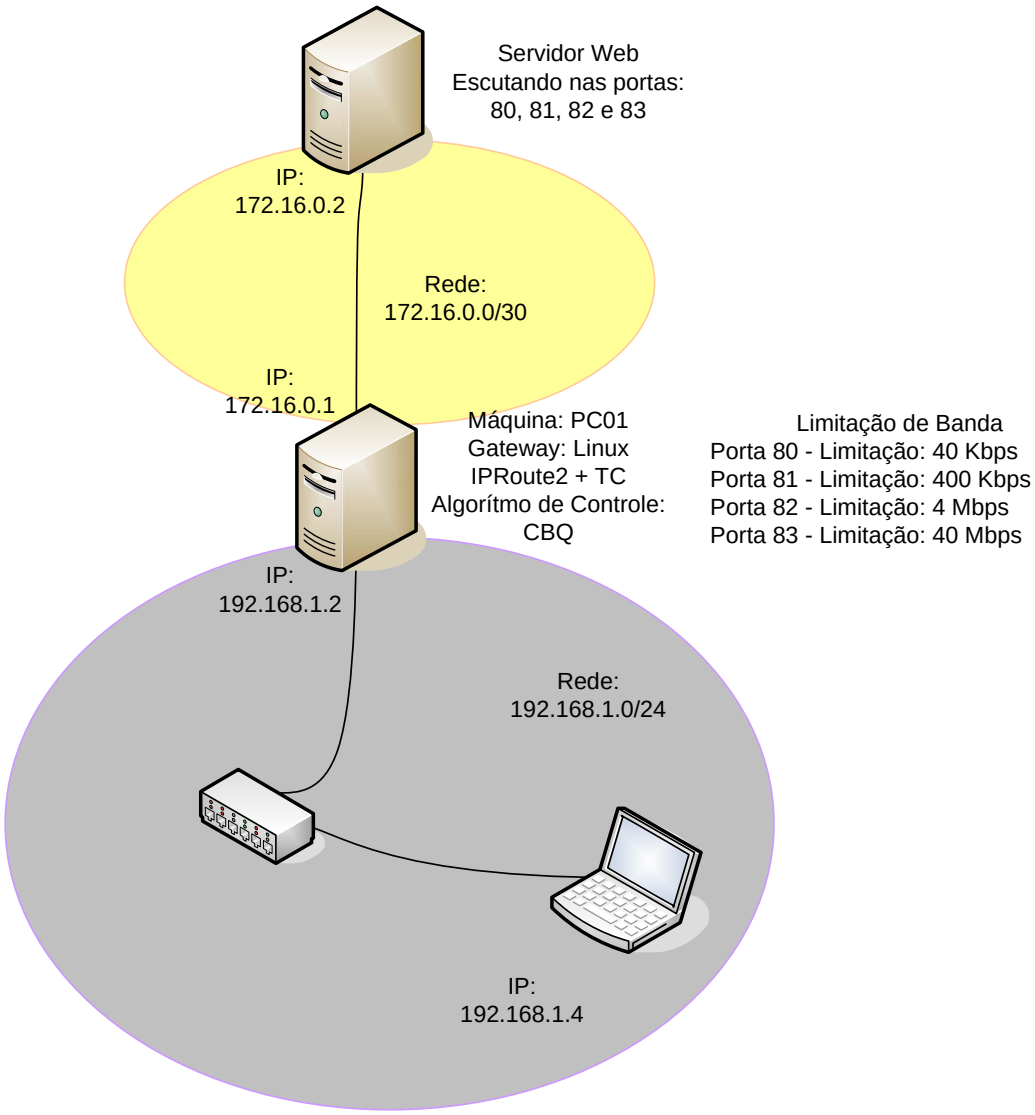


**Figura 4.2 – Diagrama de Testes - Linux**

Utilizou-se para download, tanto na solução baseada em Linux como na solução baseada em OpenBSD, um arquivo chamado “teste.tar”, conforme já mencionado.

Esse arquivo possui exatos 663152640 bytes (633 megabytes). Foi utilizado um arquivo grande, pois como a banda mais alta definida foi de 4 Mbps, era necessário que o download não terminasse antes das medidas de desempenho serem feitas.

Os downloads foram feitos, em cada porta, como demonstram as Figuras 4.3, 4.4, 4.5 e 4.6:


**Figura 4.3 – Download na porta 80 (Linux)**

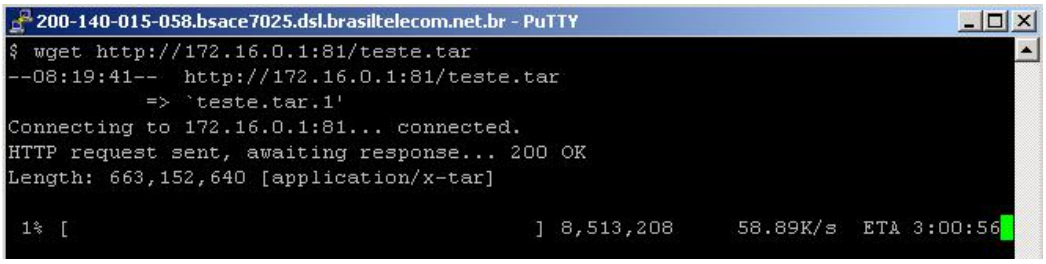


**Figura 4.4 – Download na porta 81 (Linux)**

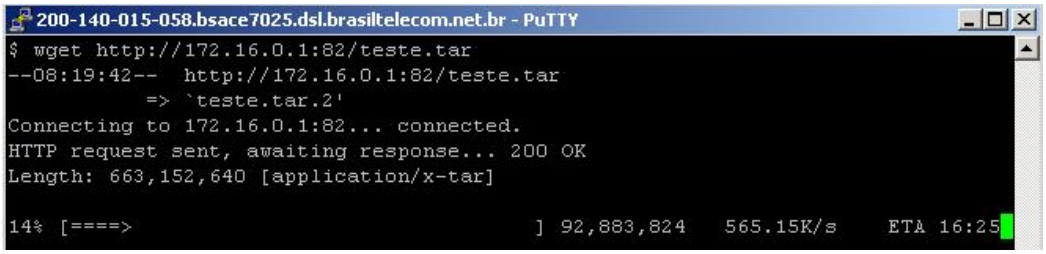


**Figura 4.5 – Download na porta 82 (Linux)**

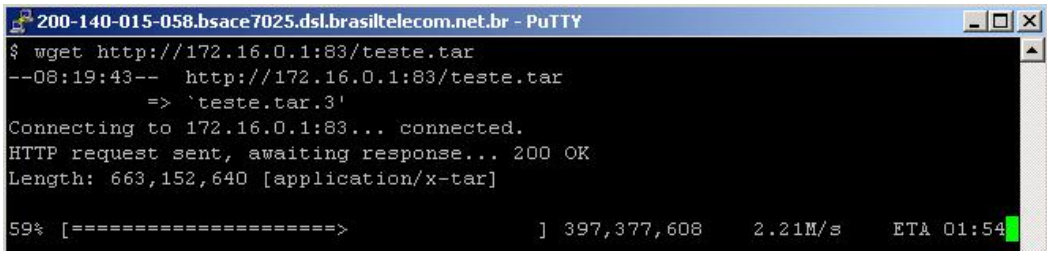


**Figura 4.6 – Download na porta 83 (Linux)**

Conforme demonstram as figuras acima, o aplicativo utilizado para realizar o download foi o “wget” [Free Software Foundation, 2005], pois esse aplicativo não guarda nada em diretórios temporários (no caso, /tmp) e faz o download diretamente para o diretório definido pelo comando. Caso não seja definido pelo comando, o download é feito para o diretório corrente.

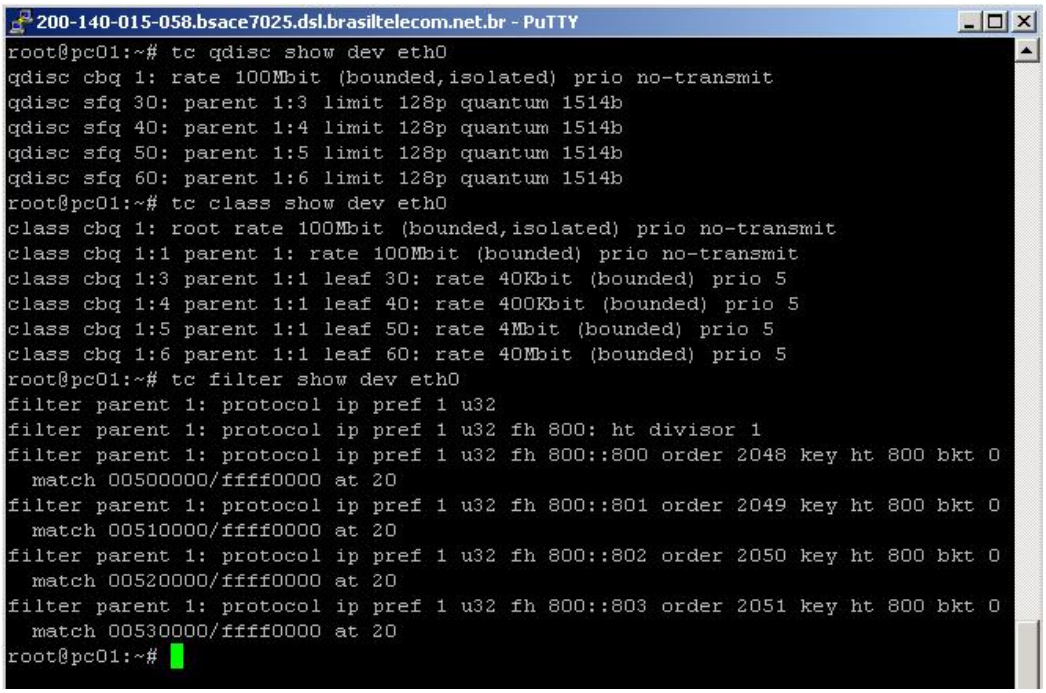


**Figura 4.7 – Qdiscs, Classes e Filtros**

A Figura 4.7 demonstra alguns dos comandos utilizados para verificar que as qdiscs, as classes e os filtros foram criados de forma correta. Nas seções seguintes, mostram-se os resultados das medidas para os dois kernels utilizados.

4.6.1 Kernel Padrão

Para medir o consumo de CPU foi utilizado o comando “top” [Linux Man, 2002] nativo do próprio sistema. A tabela abaixo mostra os valores de uso de CPU (em porcentagem) adquiridos no primeiro minuto de downloads:

| Tempo | CPU(%) |
|---|---|
| 15 | 1,0 |
| 17 | 1,0 |
| 19 | 1,0 |
| 21 | 1,5 |
| 23 | 1,0 |
| 25 | 1,0 |
| 27 | 1,5 |
| 29 | 1,0 |
| 31 | 1,0 |
| 33 | 1,0 |
| 35 | 1,0 |
| 37 | 1,5 |
| 39 | 1,0 |
| 41 | 1,0 |
| 43 | 1,0 |
| 45 | 1,0 |
| 47 | 1,5 |
| 49 | 1,0 |
| 51 | 1,0 |
| 53 | 1,0 |

**Tabela 4.2 – Medidas de desempenho (Linux – Kernel Padrão)**

O gráfico abaixo representa a Tabela 4.2 de uma forma mais legível.

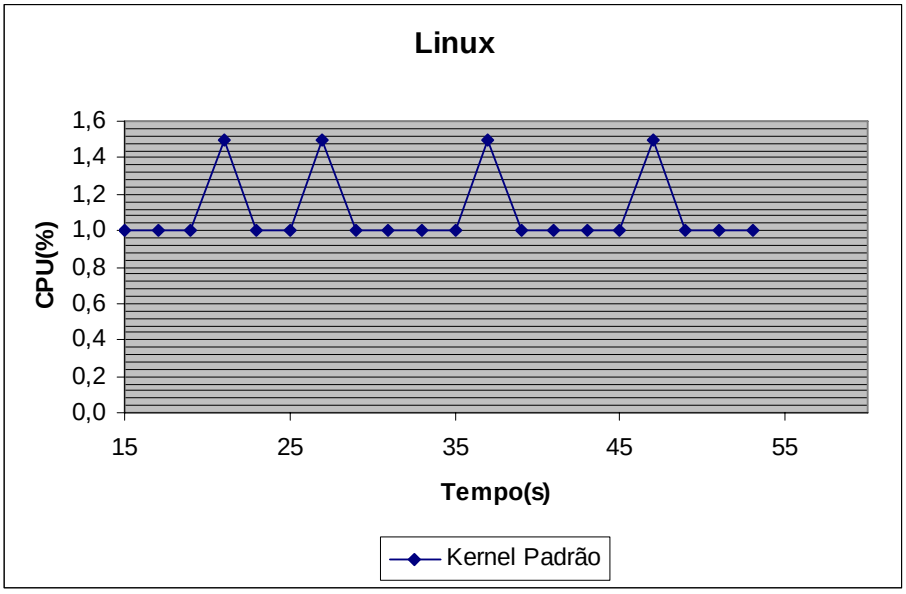


**Gráfico 4.1 – Medidas de desempenho (Linux – Kernel Padrão)**

É importante citar que, com o sistema parado, totalmente ocioso, sem estar gerenciando os downloads, a porcentagem de uso de CPU fica linearmente em 0.5% com o kernel padrão.

Pelo gráfico acima, pode-se notar que o sistema tem pequenos picos de utilização de CPU enquanto os downloads são feitos. Dessa forma, vê-se que o Linux gasta muito pouco processamento com a tarefa de enfileirar tráfego TCP/IP e limitar banda.

### 4.6.2 Kernel Customizado

A tabela abaixo, mostra o consumo de CPU (em porcentagem), no primeiro minuto de downloads, com o kernel customizado:

| Tempo | CPU(%) |
|---|---|
| 15 | 0,5 |
| 17 | 0,0 |
| 19 | 0,5 |
| 21 | 0,0 |
| 23 | 0,5 |
| 25 | 0,0 |
| 27 | 0,5 |
| 29 | 0,0 |
| 31 | 1,0 |
| 33 | 0,5 |
| 35 | 0,0 |
| 37 | 1,0 |
| 39 | 0,0 |
| 41 | 0,5 |
| 43 | 0,0 |
| 45 | 0,5 |
| 47 | 0,0 |
| 49 | 0,5 |
| 51 | 0,0 |
| 53 | 0,5 |

**Tabela 4.3 - Medidas de desempenho (Linux – Kernel Customizado)**

A tabela 4.3 é mostrada de uma forma gráfica na tabela abaixo.

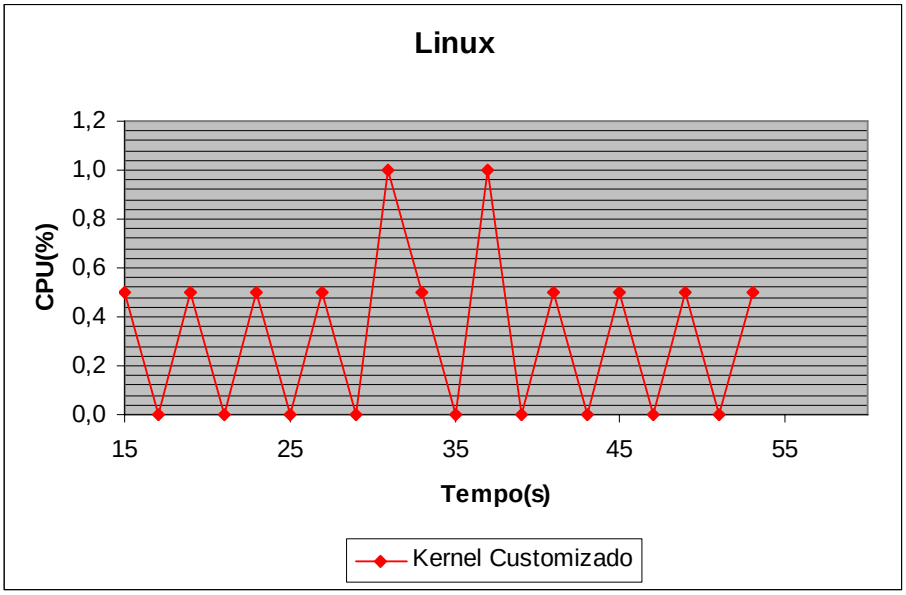


**Gráfico 4.2 - Medidas de desempenho (Linux – Kernel Customizado)**

Com o kernel customizado e sem downloads para gerenciar, o uso de CPU fica em 0.0%.

Com os downloads sendo efetuados e com o sistema gerenciando as filas, o sistema comporta-se como no gráfico acima, apresentando picos máximos de 1.0% e estando na maior parte do tempo em 0.5% de processamento.

## 4.7 OpenBSD: - Testes e Resultados

A topologia definida para os testes em OpenBSD é idêntica à do Linux, mudando-se, obviamente, os endereços IPs da rede:

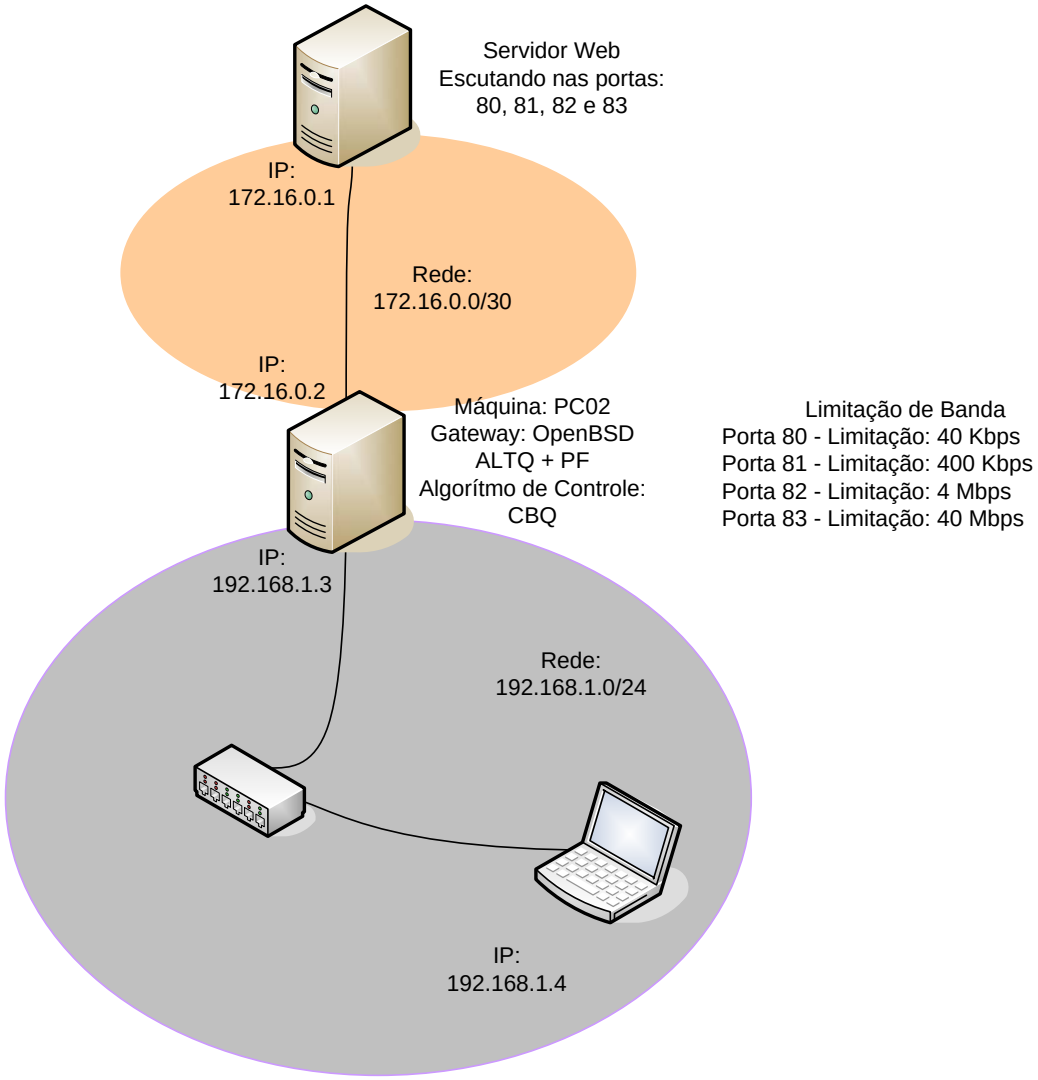


**Figura 4.8 – Diagrama de Testes - OpenBSD**

As Figuras 4.9, 4.10, 4.11 e 4.12 demonstram os downloads feitos. O mesmo arquivo de testes foi usado:

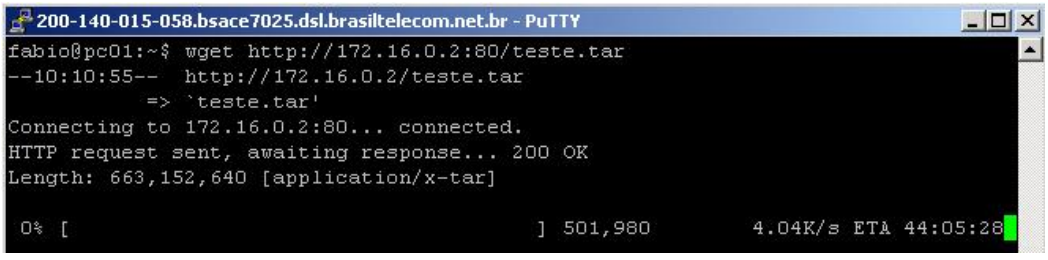


**Figura 4.9 – Download na porta 80 (OpenBSD)**

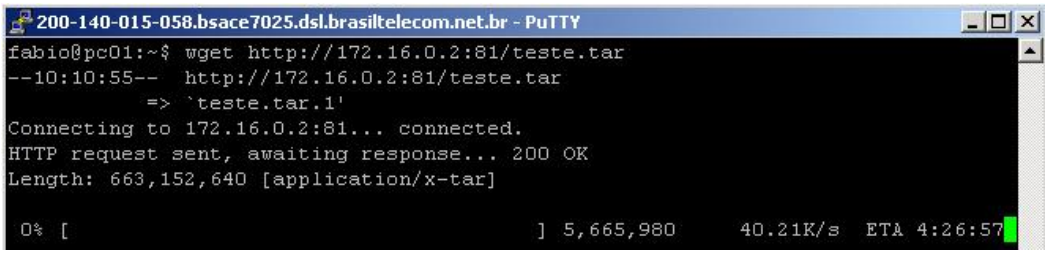


**Figura 4.10 - Download na porta 81 (OpenBSD)**

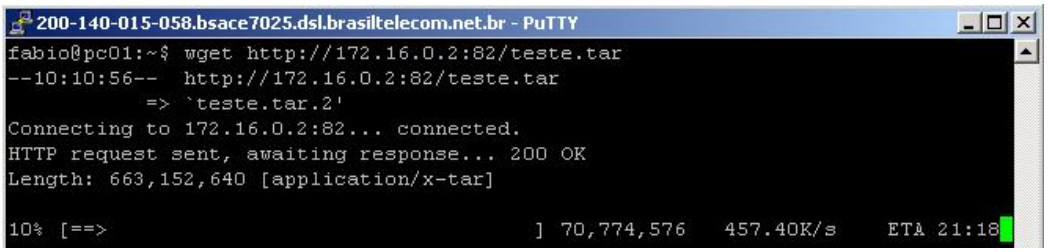


**Figura 4.11 - Download na porta 82 (OpenBSD)**

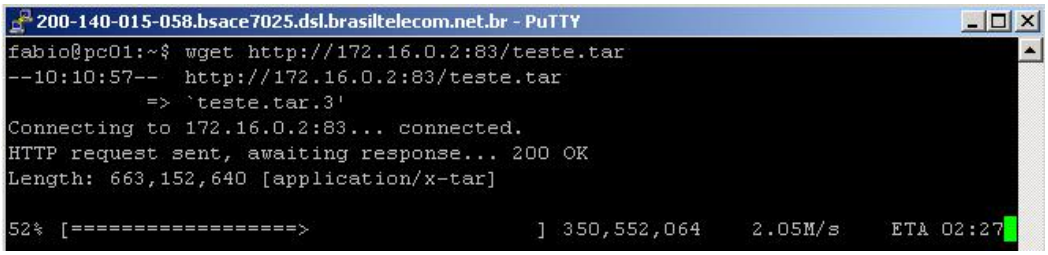


**Figura 4.12 - Download na porta 83 (OpenBSD)**

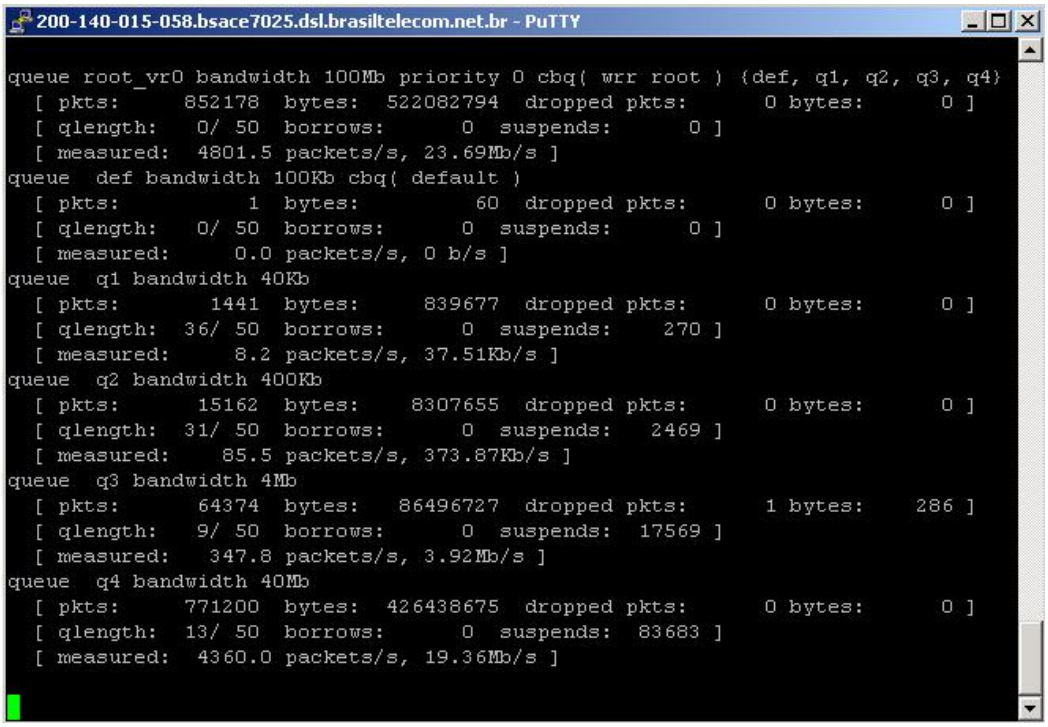


**Figura 4.13 – Filas mostradas dinamicamente com o pf**

O pf, ao contrário do tc, tem uma opção para mostrar dinamicamente as filas alocadas, conforme é demonstrado na Figura 4.13, com o comando “pfctl” [OpenBSD Man, 2002]. Os resultados das medidas são demonstrados nas próximas duas sessões.

### 4.7.1 Kernel Padrão

Conforme foi feito com o Linux, também foi medida a utilização de CPU no OpenBSD, utilizando o “top” [OpenBSD Man, 1997], conforme mostra a tabela abaixo:

| Tempo | CPU(%) |
|---|---|
| 15 | 17,5 |
| 17 | 17,1 |
| 19 | 18,3 |
| 21 | 17,1 |
| 23 | 19,4 |
| 25 | 18,4 |
| 27 | 16,7 |
| 29 | 16,7 |
| 31 | 19,1 |
| 33 | 17,5 |
| 35 | 18,3 |
| 37 | 18,2 |
| 39 | 18,7 |
| 41 | 17,6 |
| 43 | 17,4 |
| 45 | 20,2 |
| 47 | 20,6 |
| 49 | 19,4 |
| 51 | 17,8 |
| 53 | 18,7 |

**Tabela 4.4 - Medidas de desempenho (OpenBSD – Kernel Padrão)**

O gráfico abaixo demonstra de forma mais clara a utilização da CPU:

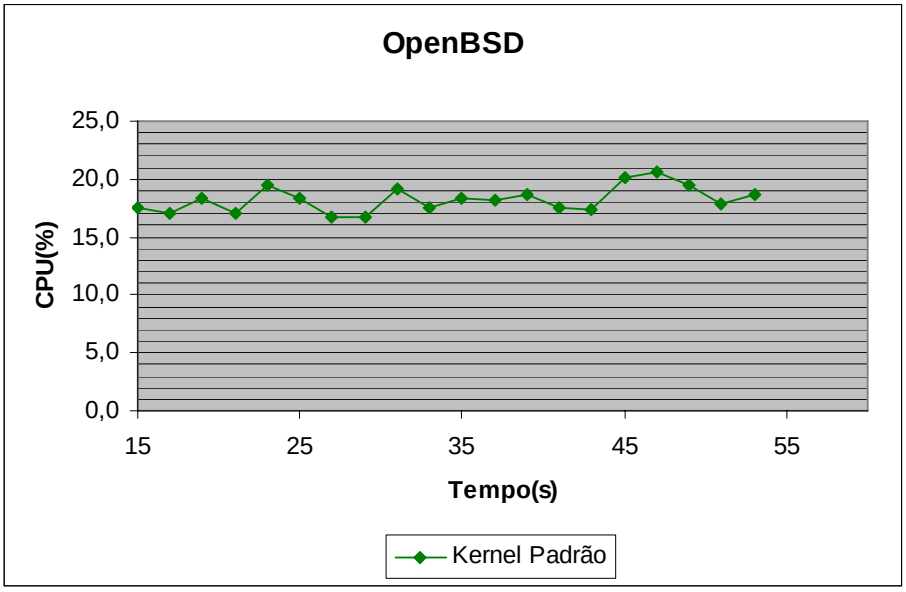


**Gráfico 4.3 - Medidas de desempenho (OpenBSD – Kernel Padrão)**

Nesse ponto é importante citar que, como no Linux, o OpenBSD quando ocioso, demonstrava valores mais baixos: mesmo com o kernel padrão do sistema, permanecia linearmente em 0.0% de utilização de CPU.

Pode-se ver claramente que a utilização de CPU no OpenBSD para o gerenciamento de filas é muito maior do que no Linux.

4.7.2 Kernel Customizado

Com o kernel customizado, os valores de utilização de CPU estão relatados na tabela abaixo:

| Tempo | CPU(%) |
|---|---|
| 15 | 16,3 |
| 17 | 16,7 |
| 19 | 17,1 |
| 21 | 16,3 |
| 23 | 15,6 |
| 25 | 17,1 |
| 27 | 13,6 |
| 29 | 16,3 |
| 31 | 17,1 |
| 33 | 16,3 |
| 35 | 18,7 |
| 37 | 18,3 |
| 39 | 15,1 |
| 41 | 12,8 |
| 43 | 11,7 |
| 45 | 12,5 |
| 47 | 15,1 |
| 49 | 14,8 |
| 51 | 17,9 |
| 53 | 14,4 |

**Tabela 4.5 - Medidas de desempenho (OpenBSD – Kernel Customizado)**

Os valores da Tabela 4.5 são mostrados de uma forma gráfica abaixo:

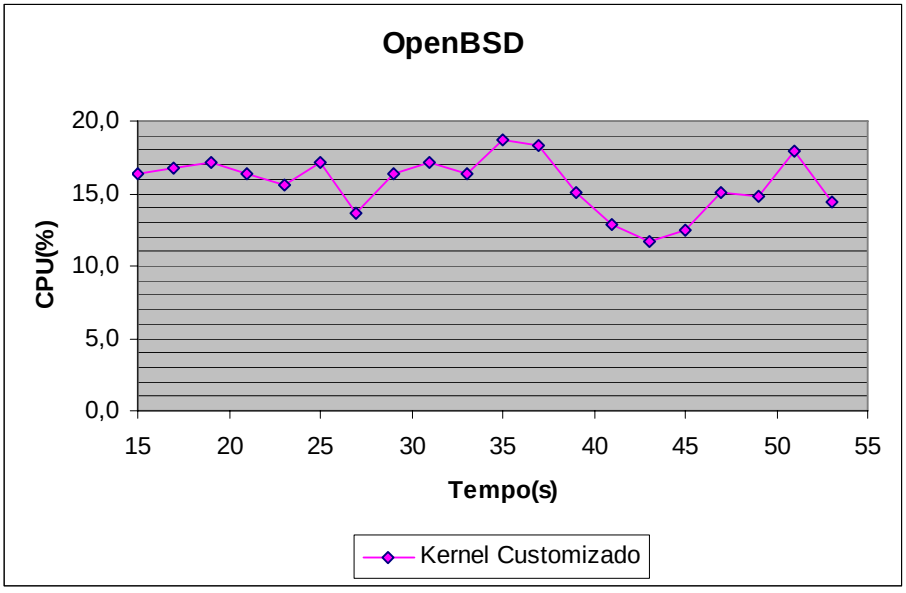


**Gráfico 4.4 - Medidas de desempenho (OpenBSD – Kernel Customizado)**

De acordo com o gráfico 4.4, pode-se ver que o OpenBSD com o kernel customizado, utiliza menos CPU do que com o kernel padrão. Porém, mesmo assim, ele continua utilizando muito mais processamento para gerenciar as filas do que o Linux.

## 4.8 Comparativo: Linux

Comparando a solução Linux, com os dois kernels, tem-se a seguinte tabela:

| Tempo(s) | Kernel Padrão | Kernel Customizado |
|---|---|---|
| 15 - 53 | CPU(%) | CPU(%) |
| 15 | 1,0 | 0,5 |
| 17 | 1,0 | 0,0 |
| 19 | 1,0 | 0,5 |
| 21 | 1,5 | 0,0 |
| 23 | 1,0 | 0,5 |
| 25 | 1,0 | 0,0 |
| 27 | 1,5 | 0,5 |
| 29 | 1,0 | 0,0 |
| 31 | 1,0 | 1,0 |
| 33 | 1,0 | 0,5 |
| 35 | 1,0 | 0,0 |
| 37 | 1,5 | 1,0 |
| 39 | 1,0 | 0,0 |
| 41 | 1,0 | 0,5 |
| 43 | 1,0 | 0,0 |
| 45 | 1,0 | 0,5 |
| 47 | 1,5 | 0,0 |
| 49 | 1,0 | 0,5 |
| 51 | 1,0 | 0,0 |
| 53 | 1,0 | 0,5 |

**Tabela 4.6 – Comparativo Linux**

Graficamente, tem-se:

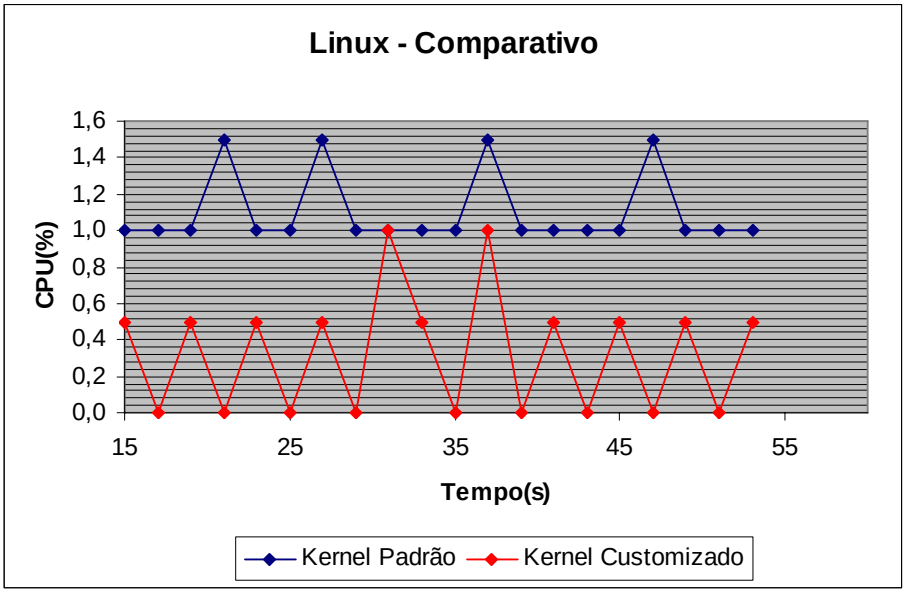


**Gráfico 4.5 – Comparativo Linux**

Apesar do intervalo de medida de CPU no Linux ser mínimo, é possível perceber que houve um ganho de desempenho com o kernel customizado. O kernel padrão, por utilizar vários recursos que nem sempre são necessários em todos os sistemas, precisar carregar, durante o processo de inicialização da máquina, vários módulos para prover a infraestrutura necessária ao funcionamento de vários periféricos ou protocolos, como Bluetooth e IPX, conforme já mencionado. Já com o kernel customizado, foi possível fazer com que o kernel só carregue em memória o que for realmente necessário para a utilização do sistema como uma solução de controle de banda.

## 4.9 Comparativo: OpenBSD

Comparando-se o OpenBSD, com os kernels padrão e customizado, tem-se:

| Tempo(s) | Kernel Padrão | Kernel Customizado |
|---|---|---|
| 15 - 53 | CPU(%) | CPU(%) |
| 15 | 17,5 | 16,3 |
| 17 | 17,1 | 16,7 |
| 19 | 18,3 | 17,1 |
| 21 | 17,1 | 16,3 |
| 23 | 19,4 | 15,6 |
| 25 | 18,4 | 17,1 |
| 27 | 16,7 | 13,6 |
| 29 | 16,7 | 16,3 |
| 31 | 19,1 | 17,1 |
| 33 | 17,5 | 16,3 |
| 35 | 18,3 | 18,7 |
| 37 | 18,2 | 18,3 |
| 39 | 18,7 | 15,1 |
| 41 | 17,6 | 12,8 |
| 43 | 17,4 | 11,7 |
| 45 | 20,2 | 12,5 |
| 47 | 20,6 | 15,1 |
| 49 | 19,4 | 14,8 |
| 51 | 17,8 | 17,9 |
| 53 | 18,7 | 14,4 |

**Tabela 4.7 – Comparativo OpenBSD**

Graficamente, tem-se:

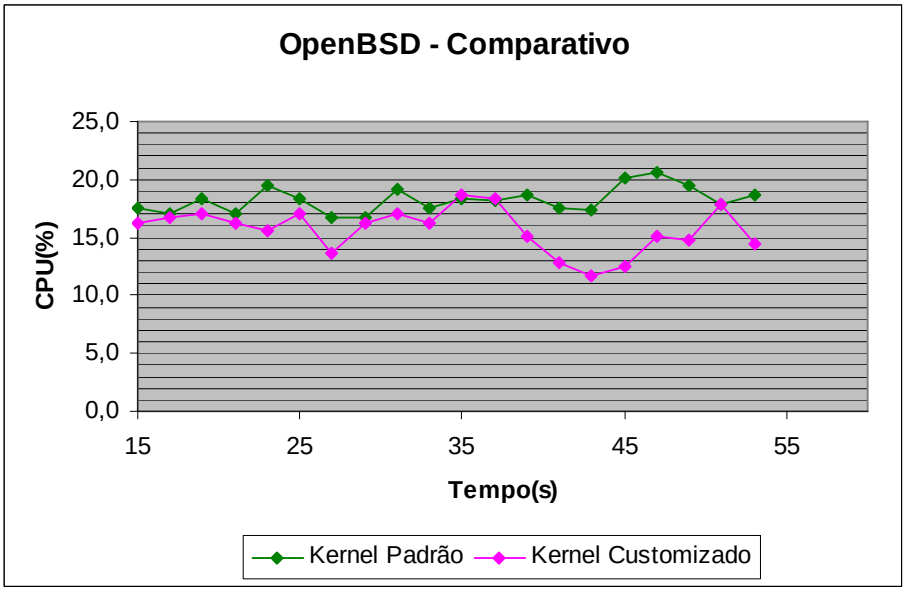


**Gráfico 4.6 – Comparativo OpenBSD**

O OpenBSD, apesar de demonstrar mais processamento do que o Linux na utilização como solução de controle de banda, também demonstra um ganho de desempenho ao se customizar o kernel.

Assim como no Linux, o kernel padrão do OpenBSD também carrega em memória vários drivers ou suportes a protocolos que são desnecessários no sistema implementado. Dessa forma, com o kernel customizado, a solução tem tarefas específicas e pode-se utilizar de um kernel também específico para executá-las.

## 4.10 Comparativo das duas soluções

Comparando-se os quatro testes feitos, tem-se a tabela abaixo:

| Tempo(s) | Kernel Padrão | Kernel Customizado | Kernel Padrão | Kernel Customizado |
|---|---|---|---|---|
| 15 - 53 | CPU(%) | CPU(%) | CPU(%) | CPU(%) |
| 15 | 17,5 | 16,3 | 1,0 | 0,5 |
| 17 | 17,1 | 16,7 | 1,0 | 0,0 |
| 19 | 18,3 | 17,1 | 1,0 | 0,5 |
| 21 | 17,1 | 16,3 | 1,5 | 0,0 |
| 23 | 19,4 | 15,6 | 1,0 | 0,5 |
| 25 | 18,4 | 17,1 | 1,0 | 0,0 |
| 27 | 16,7 | 13,6 | 1,5 | 0,5 |
| 29 | 16,7 | 16,3 | 1,0 | 0,0 |
| 31 | 19,1 | 17,1 | 1,0 | 1,0 |
| 33 | 17,5 | 16,3 | 1,0 | 0,5 |
| 35 | 18,3 | 18,7 | 1,0 | 0,0 |
| 37 | 18,2 | 18,3 | 1,5 | 1,0 |
| 39 | 18,7 | 15,1 | 1,0 | 0,0 |
| 41 | 17,6 | 12,8 | 1,0 | 0,5 |
| 43 | 17,4 | 11,7 | 1,0 | 0,0 |
| 45 | 20,2 | 12,5 | 1,0 | 0,5 |
| 47 | 20,6 | 15,1 | 1,5 | 0,0 |
| 49 | 19,4 | 14,8 | 1,0 | 0,5 |
| 51 | 17,8 | 17,9 | 1,0 | 0,0 |
| 53 | 18,7 | 14,4 | 1,0 | 0,5 |

**Tabela 4.8 – Comparativo: Linux x OpenBSD**

Graficamente, pode-se ver a diferença de uma forma mais clara:

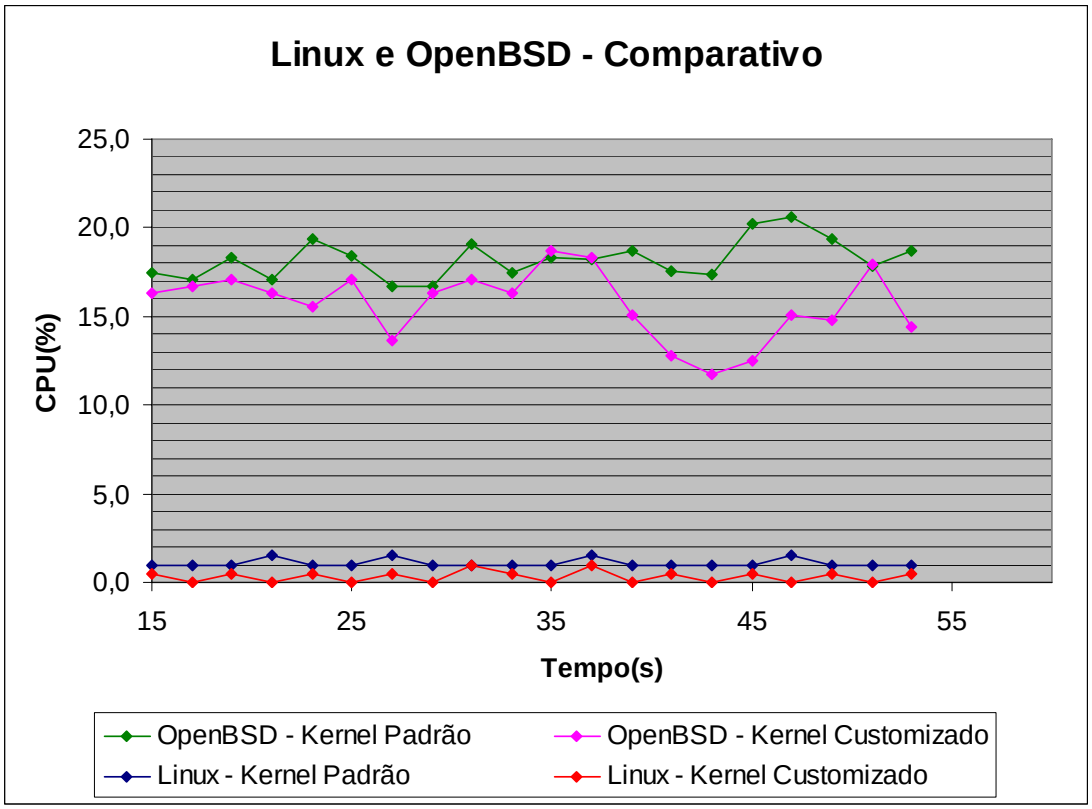


**Gráfico 4.7 – Comparativo: Linux x OpenBSD**

Pelo gráfico, é impossível deixar de notar que o Linux como solução de controle de banda tem um desempenho superior ao OpenBSD, visto que para a mesma tarefa, o primeiro utiliza muito menos recursos de máquina do que o segundo.

Além disso, nas duas soluções, foi possível aumentar sua eficiência fazendo-se uma otimização do kernel e configurando os dois sistemas para que trabalhassem somente como gateways fazendo QoS em camada 4.

Dessa forma, na análise de desempenho, o Linux demonstrou que sua implementação do algoritmo CBQ consegue ser mais eficiente do que a do OpenBSD, apesar das dificuldades de implementação do primeiro serem maiores.

## 4.11 Resultados

Conforme visto no último capítulo, o Linux demonstrou que para fazer controle de banda, utiliza muito menos processamento do que o OpenBSD.

Na primeira solução, baseada em Slackware Linux, ao se iniciar o sistema com o kernel padrão, a utilização de CPU fica em torno de 1%, e a utilização de memória RAM fica em torno de 25 megabytes. Ao se iniciar os downloads, ou seja, ao se iniciar as tarefas que devem ser processadas pela solução, a utilização de CPU aumentou em 0.5%, enquanto que a utilização de memória permaneceu inalterada. Com o kernel customizado, ao se iniciar o sistema, a utilização de CPU fica em torno de 0.0%, ou seja, tão pequena que não podia ser medida, pois a escala que o Linux lê o processamento é de 0.5% em 0.5% de processamento. Ao se iniciar os downloads, a média de

trabalho foi de 0.5% de utilização de CPU tendo alguns poucos picos de 1%. A memória RAM com o kernel customizado ficou em torno de 20 megabytes, e permaneceu inalterada enquanto os downloads eram feitos.

Na segunda solução, baseada em OpenBSD, ao se iniciar o sistema com o kernel padrão, a utilização de CPU ficava em torno de 0.0%, ou seja, o sistema totalmente ocioso. A quantidade de memória RAM utilizada é muito menor do que no Linux, apresentando apenas 6 megabytes de memória RAM utilizada. Porém, assim como no Linux, a quantidade de memória não se alterou com o processamento de filas. Já a utilização de CPU aumentou bastante, chegando a ficar, em média em 18% de utilização de CPU. Já com o kernel customizado, a média ficou em cerca de 17%, pouco abaixo da utilização com o kernel padrão. A memória RAM ficou em aproximadamente 5.5 megabytes de utilização não se alterando também com o processamento das filas. Com o sistema completamente ocioso, as medidas de processamento também marcavam 0.0% de utilização.

Nos dois sistemas, não houve alteração alguma nos acessos a disco, pois o único momento em que os discos são lidos é quando as regras ou scripts são carregadas. Após isso, tudo fica em memória RAM e não há mais acesso a disco.

Dessa forma, fica claro que a infraestrutura para QoS do Linux é mais eficiente do que a do OpenBSD, visto que, de acordo com os dados coletados, o processamento é muito maior nesse segundo sistema.

# 5. Conclusões

Um dos objetivos da engenharia é fazer um trabalho, com maior eficiência possível e utilizando o custo mais baixo possível. Dessa forma, esse projeto demonstra que é possível utilizar sistemas operacionais baseados em software livre para se fazer controle de banda em gateways de internet.

O baixo custo provém não só do fato do software em si não causar ônus, mas também do fato desses tipos de software serem feitos para um grande número de plataformas diferentes, permitindo que se instale um gateway avançado com controle de banda utilizando vários recursos com as mais novas tecnologias em uma máquina de baixo custo, podendo ser montado com peças baratas, ou até mesmo se reutilizar um computador antigo para se fazer esse tipo de serviço.

O software livre também tem a vantagem de ser extremamente flexível. Como foi demonstrado nesse trabalho, em ambas as soluções apresentadas, os códigos-fonte dos dois sistemas estão disponíveis para download na internet, e ambos foram utilizados para se recompilar os kernels dos sistemas de forma a eliminar quaisquer funções agregadas ao kernel que não fossem ter utilidade nas soluções apresentadas. Essa customização de kernel permite sempre, por mínimo que seja, um ganho de desempenho em relação aos kernels que vem por padrão no sistema.

Principalmente se o sistema for utilizado em máquinas mais antigas, e não em máquinas novas como foi mostrado nesse projeto, a customização de kernel terá uma diferença de desempenho maior ainda do que a apresentada, visto que o poder de processamento dessas primeiras é bem menor do que o dessas últimas.

Em relação às duas soluções, percebeu-se que o OpenBSD exige mais do hardware do que o Linux para a tarefa de controle de banda. Em contrapartida a esse ponto de extrema importância, deve-se enfatizar que as dificuldades de implementação encontradas no Linux são extremamente maiores do que no OpenBSD.

O Linux só permite a utilização de enfileiradores, como o CBQ, através do comando “tc” e tendo as opções de roteamento avançado, além do algoritmo correspondente habilitadas no kernel. Dessa forma, a sintaxe do comando “tc” para se fazer controle de banda é extremamente complexa e com uma quantidade muito grande de parâmetros.

Já o OpenBSD, para fazer controle de banda, utiliza a framework ALTQ, que por sua vez foi integrada ao firewall do sistema. Dessa forma, para se utilizar dos recursos de QoS do OpenBSD é necessário apenas conhecer a sintaxe da linguagem do firewall “pf”. Tendo esse conhecimento, utiliza-se o próprio firewall para se redirecionar os tráfegos para as filas definidas.

Uma das maiores dificuldades encontradas no projeto foi a dificuldade utilização dos comandos para controle de banda no Linux em relação ao OpenBSD. Ao se entender a sintaxe utilizada pelo firewall do OpenBSD (PF), o entendimento da utilização de suas filas se tornou muito mais simples. De modo contrário, quando se iniciou a implementação da solução em Linux, as regras e parâmetros do comando “tc” utilizadas para se fazer controle de banda se mostraram extremamente complexas.

Inicialmente, foram feitos testes com filas simples. Após isso, foi utilizado o algoritmo de classe PRIO, que era mais fácil de entender do que o CBQ. Porém, ao se passar a utilização desse último, a quantidade de parâmetros e regras se tornou tão complexa e praticamente ininteligível, que tornava o entendimento das regras uma tarefa extremamente difícil.

Além da dificuldade encontrada na sintaxe dos comandos do Linux, um outro ponto que exigiu um esforço demasiadamente grande foi a customização dos kernels dos dois sistemas. Como o objetivo do projeto era customizar o kernel para que o mesmo só possuísse funções que tivessem alguma relação ao trabalho que a solução iria executar, essa customização foi muito trabalhosa, pois todas as opções do kernel tiveram que ser entendidas e cuidadosamente selecionadas, para que se definisse se fariam parte do kernel customizado da solução, ou não.

A desvantagem do software livre em relação ao comercial é a dificuldade de utilização. Apesar de o OpenBSD ser mais simples do que o Linux para se implementar controle de banda, ele ainda assim é de difícil utilização, principalmente para os que não tem experiência com o sistema (OpenBSD). Dessa forma, a utilização de produtos que são controlados por linha de comando, ao invés de interfaces gráficas, é sempre vista de forma negativa por alguém que esteja precisando de uma solução como essas. Outra desvantagem é que já existem algumas soluções de controle de banda comerciais baseadas em hardware. Ou seja, ao se comprar um produto como esses, o suporte se estende ao hardware, o que não acontece com o software livre (cujo hardware será determinado por quem implementar).

Porém, o software livre no mundo já demonstra um grau de maturidade bastante elevado e já está sendo utilizado internamente em muitas empresas. Inclusive já existem vários produtos comerciais que são baseados em software livre, como é o caso do Red Hat Enterprise Linux. Os próprios códigos-fonte de algumas soluções também são usados em produtos comerciais. Por exemplo, no Microsoft Services for Unix, existe parte do código que foi retirada do OpenBSD. Dessa forma, o software livre vem ganhando maturidade suficiente para demonstrar seu poder de ser utilizado em ambientes críticos.

Com isso, esse projeto agregará bastante valor a futuros projetos ou produtos que tenham por objetivo principal o QoS, de forma a demonstrar todo o conhecimento necessário, scripts, regras e uma solução pronta para se trabalhar com controle de banda em um gateway de borda de internet utilizando software livre, além de demonstrar a potencial otimização que pode ser feita em sistemas de código-aberto com o intuito de aumentar seu desempenho.

## 5.1 Trabalhos Futuros

Existem algumas idéias que decorrem desse trabalho que podem ser utilizadas, pesquisadas e projetadas. Uma delas, é a criação de uma interface gráfica para se fazer o controle de banda dinamicamente utilizando ou o OpenBSD ou o Linux.

Outra idéia que pode ser implementada para a geração de um futuro produto de controle de banda é alterar-se a forma de funcionamento do “pf” ou do “tc” para que os filtros criados para controle de banda possam ser utilizados em camada 7, ou seja, em nível de aplicação.

Em terceiro lugar, pode-se utilizar a solução criada e adicionar a funcionalidade de “alta disponibilidade” a ela, colocando-se duas máquinas em cluster fazendo o controle de banda, e no caso de alguma delas falhar, a outra assume suas atividades.

# Referências


Cisco – “Introduction to IP QoS (Course)” – 2001

Comer, Douglas E. – “Interligação em Rede com TCP/IP” – Rio de Janeiro: Campos, 1998

Devera, Martin – “HTB Queueing Discipline Manual” – Maio de 2002 – URL: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm

Diógenes, Yuri – “Certificação Cisco” - 2002

Floyd, Sally - "Notes on CBQ and Guaranteed Service" - Julho de 1995 - URL: http://www.aciri.org/floyd/papers/guaranteed.ps

Free Software Foundation – “GNU Wget Manual“ – Novembro de 2005 – URL: http://www.gnu.org/software/wget/manual/wget.html

Hubert, Bert – “Linux Advanced Routing & Traffic Control” – Outubro de 2003 - URL: http://lartc.org/

Jha, Sanjay; Hassan, Mahbub – “Engineering Internet QoS” – 2002, Artech House, Inc.

Korff, Yanek; Hope, Paco; Potter, Bruce – “Mastering FreeBSD and OpenBSD Security” – O’Reilly Media, 2005

Lamb, Mark – “iproute2+tc notes” – 1999 – URL: http://snafu.freedom.org/linux2.2/iproute-notes.html

Linux Man – “CBQ – Class Based Queueing” – Dezembro de 2001

Linux Man – “Linux User’s Manual: top” – Setembro de 2002

Linux Man – “Manipulate Traffic Control Settings” – Dezembro de 2001

Lucas, Michael – “Absolute OpenBSD” – No Starch Press, 2003.

Lucas, Michael – “Dominando BSD” – Rio de Janeiro: Editora Ciência Moderna, 2003.

Maxwell, Scott – “Kernel do Linux” – São Paulo: Makron Books, 2000.

McClure, Stuart; Scambray, Joel; Kurtz, George – “Hackers Expostos” – Rio de Janeiro: 2003, Elsevier Editora Ltda.

Michael, Randal K. – “Dominando Unix Shell Scripting” – Rio de Janeiro: Elsevier, 2003.

OpenBSD Archives – “OpenBSD Firewall Minimum hardware requirements” – Julho de 2000 – URL: http://www.monkey.org/openbsd/archive/misc/0007/msg01459.html

OpenBSD FAQ – “Documentation and Frequently Asked Questions” – Novembro de 2005 – URL: http://www.openbsd.org/faq/index.html

OpenBSD Man - “OpenBSD Reference Manual: top” – Agosto de 1997

OpenBSD Man – “Packet Filter Configuration File” – Novembro de 2002

OpenBSD Man – “PF – Packet Filter” – Junho de 2004 /

OpenBSD Man – “pfctl Man Page” – Novembro de 2002

OpenBSD PF FAQ – “The OpenBSD Packet Filter” – Novembro de 2005 - URL: http://www.openbsd.org/faq/pf

OpenBSD/i386 – “Supported hardware” – Novembro de 2005 – URL: http://www.openbsd.org/i386.html

Packeteer – “PacketShaper Datasheet” – 2005 – URL: http://www.packeteer.com

RFC1349 – “Type of Service in the Internet Protocol Suite” – Julho de 1992 – URL: http://www.ietf.org/rfc/rfc1349.txt

RFC1918 – “Address Allocation for Private Internets” – Fevereiro de 1996 –URL: http://www.geektools.com/rfc/rfc1918.txt

RFC2001 – “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms” – Janeiro de 1997 – URL: http://rfc.sunsite.dk/rfc/rfc2001.html

RFC2309 – “Recommendations on Queue Management and Congestion Avoidance in the Internet” – Abril de 1998 – URL: http://www.faqs.org/rfcs/rfc2309.html

RFC3168 – “The Addition of Explicit Congestion Notification (ECN) to IP” – Setembro de 2001 – URL: http://www.rfc-editor.org/rfc/rfc3168.txt

Rizzo, Luigi – “Dummynet” – 2000 – URL: http://info.iet.unipi.it/~luigi/ip_dummynet/

Stevens, W. Richard - "TCP/IP Illustrated, Vol. 1, The Protocols" - Tucson, Arizona: , 1993.

Tanenbaum, Andrew S. – “Computer Networks” – Quarta edição, Prentice Hall, 2003.

Welsh, Matt; Kaufman, Lar – “Dominando o Linux” – Rio de Janeiro: Editora Ciência Moderna, 1997.

# Anexo 1: Kernel do Linux – Configuração

Algumas opções foram retiradas do kernel para que o sistema fosse customizado para o funcionamento como uma solução de controle de banda. Dessa forma, o arquivo de configuração do kernel foi bastante alterado. O quadro abaixo representa o arquivo **/usr/src/linux/.config** do sistema operacional Slackware Linux, com as opções do kernel que foram retiradas, para que o sistema fosse customizado.

```
CONFIG_EXPERIMENTAL=y
CONFIG_MODULES=y
CONFIG_KMOD=y
CONFIG_M486=y
CONFIG_X86_L1_CACHE_SHIFT=4
CONFIG_X86_USE_STRING_486=y
CONFIG_X86_ALIGNMENT_16=y
CONFIG_X86_PPRO_FENCE=y
CONFIG_TOSHIBA=m
CONFIG_I8K=m
CONFIG_MICROCODE=m
CONFIG_X86_MSR=m
CONFIG_X86_CPUID=m
CONFIG_EDD=m
CONFIG_MATH_EMULATION=y
CONFIG_MTRR=y
CONFIG_NET=y
CONFIG_ISA=y
CONFIG_PCI_NAMES=y
CONFIG_HOTPLUG=y
CONFIG_PCMCIA=m
CONFIG_CARDBUS=y
CONFIG_TCIC=y
CONFIG_I82092=y
CONFIG_I82365=y
CONFIG_HOTPLUG_PCI=m
CONFIG_HOTPLUG_PCI_COMPAQ=m
CONFIG_HOTPLUG_PCI_SHPC=m
CONFIG_HOTPLUG_PCI_SHPC_PHPRM_LEGACY=y
CONFIG_HOTPLUG_PCI_PCIE=m
CONFIG_SYSVIPC=y
CONFIG_BSD_PROCESS_ACCT=y
CONFIG_SYSCTL=y
CONFIG_BINFMT_AOUT=m
CONFIG_BINFMT_MISC=m
CONFIG_PM=y
CONFIG_APM=m
CONFIG_PARPORT=m
CONFIG_PARPORT_PC=m
CONFIG_PARPORT_PC_CML1=m
CONFIG_PARPORT_SERIAL=m
CONFIG_PARPORT_PC_PCMCIA=m
CONFIG_PARPORT_1284=y
CONFIG_PNP=m
CONFIG_ISAPNP=m
CONFIG_BLK_DEV_FD=y
CONFIG_BLK_DEV_XD=m
CONFIG_PARIDE=m
CONFIG_PARIDE_PARPORT=m
CONFIG_PARIDE_PD=m
CONFIG_PARIDE_PCD=m
CONFIG_PARIDE_PF=m
```

```
CONFIG_PARIDE_PT=m
CONFIG_PARIDE_PG=m
CONFIG_PARIDE_ATEN=m
CONFIG_PARIDE_BPCK=m
CONFIG_PARIDE_BPCK6=m
CONFIG_PARIDE_COMM=m
CONFIG_PARIDE_DSTR=m
CONFIG_PARIDE_FIT2=m
CONFIG_PARIDE_FIT3=m
CONFIG_PARIDE_EPAT=m
CONFIG_PARIDE_EPATC8=y
CONFIG_PARIDE_EPIA=m
CONFIG_PARIDE_FRIQ=m
CONFIG_PARIDE_FRPW=m
CONFIG_PARIDE_KBIC=m
CONFIG_PARIDE_KTTI=m
CONFIG_PARIDE_ON20=m
CONFIG_PARIDE_ON26=m
CONFIG_BLK_CPQ_DA=m
CONFIG_BLK_CPQ_CISS_DA=m
CONFIG_CISS_SCSI_TAPE=y
CONFIG_BLK_DEV_DAC960=m
CONFIG_BLK_DEV_UMEM=m
CONFIG_BLK_DEV_SX8=m
CONFIG_BLK_DEV_LOOP=y
CONFIG_BLK_DEV_NBD=m
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_SIZE=7777
CONFIG_BLK_DEV_INITRD=y
CONFIG_MD=y
CONFIG_BLK_DEV_MD=y
CONFIG_MD_LINEAR=y
CONFIG_MD_RAID0=y
CONFIG_MD_RAID1=y
CONFIG_MD_RAID5=y
CONFIG_MD_MULTIPATH=m
CONFIG_BLK_DEV_LVM=y
CONFIG_PACKET=y
CONFIG_PACKET_MMAP=y
CONFIG_NETLINK_DEV=m
CONFIG_NETFILTER=y
CONFIG_FILTER=y
CONFIG_UNIX=y
CONFIG_INET=y
CONFIG_IP_MULTICAST=y
CONFIG_IP_ADVANCED_ROUTER=y
CONFIG_IP_MULTIPLE_TABLES=y
CONFIG_IP_ROUTE_FWMARK=y
CONFIG_IP_ROUTE_NAT=y
CONFIG_IP_ROUTE_MULTIPATH=y
CONFIG_IP_ROUTE_TOS=y
CONFIG_IP_ROUTE_VERBOSE=y
CONFIG_NET_IPIP=m
CONFIG_NET_IPGRE=m
CONFIG_NET_IPGRE_BROADCAST=y
CONFIG_IP_MROUTE=y
CONFIG_IP_PIMSM_V1=y
CONFIG_SYN_COOKIES=y
CONFIG_IP_NF_CONNTRACK=m
CONFIG_IP_NF_FTP=m
CONFIG_IP_NF_AMANDA=m
CONFIG_IP_NF_TFTP=m
CONFIG_IP_NF_IRC=m
CONFIG_IP_NF_QUEUE=m
```

```
CONFIG_IP_NF_IPTABLES=m
CONFIG_IP_NF_MATCH_LIMIT=m
CONFIG_IP_NF_MATCH_MAC=m
CONFIG_IP_NF_MATCH_PKTTYPE=m
CONFIG_IP_NF_MATCH_MARK=m
CONFIG_IP_NF_MATCH_MULTIPORT=m
CONFIG_IP_NF_MATCH_TOS=m
CONFIG_IP_NF_MATCH_RECENT=m
CONFIG_IP_NF_MATCH_ECN=m
CONFIG_IP_NF_MATCH_DSCP=m
CONFIG_IP_NF_MATCH_AH_ESP=m
CONFIG_IP_NF_MATCH_LENGTH=m
CONFIG_IP_NF_MATCH_TTL=m
CONFIG_IP_NF_MATCH_TCPMSS=m
CONFIG_IP_NF_MATCH_HELPER=m
CONFIG_IP_NF_MATCH_STATE=m
CONFIG_IP_NF_MATCH_CONNTRACK=m
CONFIG_IP_NF_MATCH_UNCLEAN=m
CONFIG_IP_NF_MATCH_OWNER=m
CONFIG_IP_NF_FILTER=m
CONFIG_IP_NF_TARGET_REJECT=m
CONFIG_IP_NF_TARGET_MIRROR=m
CONFIG_IP_NF_NAT=m
CONFIG_IP_NF_NAT_NEEDED=y
CONFIG_IP_NF_TARGET_MASQUERADE=m
CONFIG_IP_NF_TARGET_REDIRECT=m
CONFIG_IP_NF_NAT_AMANDA=m
CONFIG_IP_NF_NAT_SNMP_BASIC=m
CONFIG_IP_NF_NAT_IRC=m
CONFIG_IP_NF_NAT_FTP=m
CONFIG_IP_NF_NAT_TFTP=m
CONFIG_IP_NF_MANGLE=m
CONFIG_IP_NF_TARGET_TOS=m
CONFIG_IP_NF_TARGET_ECN=m
CONFIG_IP_NF_TARGET_DSCP=m
CONFIG_IP_NF_TARGET_MARK=m
CONFIG_IP_NF_TARGET_LOG=m
CONFIG_IP_NF_TARGET_ULOG=m
CONFIG_IP_NF_TARGET_TCPMSS=m
CONFIG_IP_NF_ARPTABLES=m
CONFIG_IP_NF_ARPFILTER=m
CONFIG_IP_NF_ARP_MANGLE=m
CONFIG_IP_NF_COMPAT_IPCHAINS=m
CONFIG_IP_NF_NAT_NEEDED=y
CONFIG_IP_VS=m
CONFIG_IP_VS_TAB_BITS=12
CONFIG_IP_VS_RR=m
CONFIG_IP_VS_WRR=m
CONFIG_IP_VS_LC=m
CONFIG_IP_VS_WLC=m
CONFIG_IP_VS_LBLC=m
CONFIG_IP_VS_LBLCR=m
CONFIG_IP_VS_DH=m
CONFIG_IP_VS_SH=m
CONFIG_IP_VS_SED=m
CONFIG_IP_VS_NQ=m
CONFIG_IP_VS_FTP=m
CONFIG_IPV6=m
CONFIG_IP6_NF_IPTABLES=m
CONFIG_IP6_NF_MATCH_LIMIT=m
CONFIG_IP6_NF_MATCH_MAC=m
CONFIG_IP6_NF_MATCH_RT=m
CONFIG_IP6_NF_MATCH_OPTS=m
CONFIG_IP6_NF_MATCH_FRAG=m
```

```
CONFIG_IP6_NF_MATCH_HL=m
CONFIG_IP6_NF_MATCH_MULTIPORT=m
CONFIG_IP6_NF_MATCH_OWNER=m
CONFIG_IP6_NF_MATCH_MARK=m
CONFIG_IP6_NF_MATCH_IPV6HEADER=m
CONFIG_IP6_NF_MATCH_AHESP=m
CONFIG_IP6_NF_MATCH_LENGTH=m
CONFIG_IP6_NF_MATCH_EUI64=m
CONFIG_IP6_NF_FILTER=m
CONFIG_IP6_NF_TARGET_LOG=m
CONFIG_IP6_NF_MANGLE=m
CONFIG_IP6_NF_TARGET_MARK=m
CONFIG_KHTTPD=m
CONFIG_IP_SCTP=m
CONFIG_SCTP_HMAC_MD5=y
CONFIG_VLAN_8021Q=m
CONFIG_IPX=m
CONFIG_ATALK=m
CONFIG_DECNET=m
CONFIG_BRIDGE=m
CONFIG_X25=m
CONFIG_LAPB=m
CONFIG_WAN_ROUTER=m
CONFIG_NET_SCHED=y
CONFIG_NET_SCH_CBQ=m
CONFIG_NET_SCH_HTB=m
CONFIG_NET_SCH_CSZ=m
CONFIG_NET_SCH_HFSC=m
CONFIG_NET_SCH_PRIO=m
CONFIG_NET_SCH_RED=m
CONFIG_NET_SCH_SFQ=m
CONFIG_NET_SCH_TEQL=m
CONFIG_NET_SCH_TBF=m
CONFIG_NET_SCH_GRED=m
CONFIG_NET_SCH_NETEM=m
CONFIG_NET_SCH_DSMARK=m
CONFIG_NET_SCH_INGRESS=m
CONFIG_NET_QOS=y
CONFIG_NET_ESTIMATOR=y
CONFIG_NET_CLS=y
CONFIG_NET_CLS_TCINDEX=m
CONFIG_NET_CLS_ROUTE4=m
CONFIG_NET_CLS_ROUTE=y
CONFIG_NET_CLS_FW=m
CONFIG_NET_CLS_U32=m
CONFIG_NET_CLS_RSVP=m
CONFIG_NET_CLS_RSVP6=m
CONFIG_NET_CLS_POLICE=y
CONFIG_NET_PKTGEN=m
CONFIG_PHONE=m
CONFIG_PHONE_IXJ=m
CONFIG_PHONE_IXJ_PCMCIA=m
CONFIG_BLK_DEV_IDECS=m
CONFIG_BLK_DEV_DELKIN=m
CONFIG_BLK_DEV_IDECD=y
CONFIG_BLK_DEV_IDETAPE=m
CONFIG_BLK_DEV_IDEFLOPPY=y
CONFIG_BLK_DEV_IDESCSI=m
CONFIG_BLK_DEV_CMD640=y
CONFIG_BLK_DEV_GENERIC=y
CONFIG_IDEPCI_SHARE_IRQ=y
CONFIG_BLK_DEV_IDEDMA_PCI=y
CONFIG_IDEDMA_PCI_AUTO=y
CONFIG_BLK_DEV_IDEDMA=y
```

```
CONFIG_BLK_DEV_ADMA100=y
CONFIG_BLK_DEV_AEC62XX=y
CONFIG_BLK_DEV_ALI15X3=y
CONFIG_BLK_DEV_AMD74XX=y
CONFIG_BLK_DEV_ATIIXP=y
CONFIG_BLK_DEV_CMD64X=y
CONFIG_BLK_DEV_TRIFLEX=y
CONFIG_BLK_DEV_CS5530=y
CONFIG_BLK_DEV_HPT34X=y
CONFIG_BLK_DEV_HPT366=y
CONFIG_BLK_DEV_PIIX=y
CONFIG_BLK_DEV_PDC202XX_OLD=y
CONFIG_BLK_DEV_PDC202XX_NEW=y
CONFIG_BLK_DEV_RZ1000=y
CONFIG_BLK_DEV_SC1200=y
CONFIG_BLK_DEV_SVWKS=y
CONFIG_BLK_DEV_SIIMAGE=y
CONFIG_BLK_DEV_SIS5513=y
CONFIG_BLK_DEV_SLC90E66=y
CONFIG_BLK_DEV_VIA82CXXX=y
CONFIG_IDE_CHIPSETS=y
CONFIG_BLK_DEV_4DRIVES=y
CONFIG_BLK_DEV_ALI14XX=y
CONFIG_BLK_DEV_DTC2278=y
CONFIG_BLK_DEV_HT6560B=y
CONFIG_BLK_DEV_PDC4030=y
CONFIG_BLK_DEV_QD65XX=y
CONFIG_BLK_DEV_UMC8672=y
CONFIG_IDEDMA_AUTO=y
CONFIG_BLK_DEV_PDC202XX=y
CONFIG_BLK_DEV_ATARAID=m
CONFIG_BLK_DEV_ATARAID_PDC=m
CONFIG_BLK_DEV_ATARAID_HPT=m
CONFIG_BLK_DEV_ATARAID_MEDLEY=m
CONFIG_BLK_DEV_ATARAID_SII=m
CONFIG_SCSI=y
CONFIG_BLK_DEV_SD=y
CONFIG_SD_EXTRA_DEVS=40
CONFIG_CHR_DEV_ST=m
CONFIG_CHR_DEV_OSST=m
CONFIG_BLK_DEV_SR=y
CONFIG_SR_EXTRA_DEVS=2
CONFIG_CHR_DEV_SG=y
CONFIG_SCSI_DEBUG_QUEUES=y
CONFIG_BLK_DEV_3W_XXXX_RAID=m
CONFIG_SCSI_7000FASST=m
CONFIG_SCSI_ACARD=m
CONFIG_SCSI_AHA152X=m
CONFIG_SCSI_AHA1542=m
CONFIG_SCSI_AHA1740=m
CONFIG_SCSI_AACRAID=m
CONFIG_SCSI_AIC7XXX=m
CONFIG_AIC7XXX_CMDS_PER_DEVICE=8
CONFIG_AIC7XXX_RESET_DELAY_MS=15000
CONFIG_AIC7XXX_DEBUG_MASK=0
CONFIG_SCSI_AIC79XX=m
CONFIG_AIC79XX_CMDS_PER_DEVICE=32
CONFIG_AIC79XX_RESET_DELAY_MS=15000
CONFIG_AIC79XX_DEBUG_MASK=0
CONFIG_SCSI_AIC7XXX_OLD=m
CONFIG_AIC7XXX_OLD_CMDS_PER_DEVICE=8
CONFIG_SCSI_DPT_I2O=m
CONFIG_SCSI_ADVANSYS=m
CONFIG_SCSI_IN2000=m
```

```
CONFIG_SCSI_AM53C974=m
CONFIG_SCSI_MEGARAID=m
CONFIG_SCSI_MEGARAID2=m
CONFIG_SCSI_SATA=y
CONFIG_SCSI_SATA_AHCI=m
CONFIG_SCSI_SATA_SVW=m
CONFIG_SCSI_ATA_PIIX=m
CONFIG_SCSI_SATA_NV=m
CONFIG_SCSI_SATA_PROMISE=m
CONFIG_SCSI_SATA_SX4=m
CONFIG_SCSI_SATA_SIL=m
CONFIG_SCSI_SATA_SIS=m
CONFIG_SCSI_SATA_ULI=m
CONFIG_SCSI_SATA_VIA=m
CONFIG_SCSI_SATA_VITESSE=m
CONFIG_SCSI_BUSLOGIC=m
CONFIG_SCSI_CPQFCTS=m
CONFIG_SCSI_DMX3191D=m
CONFIG_SCSI_DTC3280=m
CONFIG_SCSI_EATA=m
CONFIG_SCSI_EATA_MAX_TAGS=16
CONFIG_SCSI_EATA_DMA=m
CONFIG_SCSI_EATA_PIO=m
CONFIG_SCSI_FUTURE_DOMAIN=m
CONFIG_SCSI_GDTH=m
CONFIG_SCSI_GENERIC_NCR5380=m
CONFIG_SCSI_GENERIC_NCR53C400=y
CONFIG_SCSI_G_NCR5380_PORT=y
CONFIG_SCSI_IPS=m
CONFIG_SCSI_INITIO=m
CONFIG_SCSI_INIA100=m
CONFIG_SCSI_PPA=m
CONFIG_SCSI_IMM=m
CONFIG_SCSI_IZIP_EPP16=y
CONFIG_SCSI_NCR53C406A=m
CONFIG_SCSI_NCR53C7xx=m
CONFIG_SCSI_SYM53C8XX_2=m
CONFIG_SCSI_SYM53C8XX_DMA_ADDRESSING_MODE=1
CONFIG_SCSI_SYM53C8XX_DEFAULT_TAGS=16
CONFIG_SCSI_SYM53C8XX_MAX_TAGS=64
CONFIG_SCSI_NCR53C8XX=m
CONFIG_SCSI_SYM53C8XX=m
CONFIG_SCSI_NCR53C8XX_DEFAULT_TAGS=8
CONFIG_SCSI_NCR53C8XX_MAX_TAGS=32
CONFIG_SCSI_NCR53C8XX_SYNC=20
CONFIG_SCSI_NCR53C8XX_PQS_PDS=y
CONFIG_SCSI_PAS16=m
CONFIG_SCSI_PCI2000=m
CONFIG_SCSI_PCI2220I=m
CONFIG_SCSI_PSI240I=m
CONFIG_SCSI_QLOGIC_FAS=m
CONFIG_SCSI_QLOGIC_ISP=m
CONFIG_SCSI_QLOGIC_FC=m
CONFIG_SCSI_QLOGIC_FC_FIRMWARE=y
CONFIG_SCSI_QLOGIC_1280=m
CONFIG_SCSI_SEAGATE=m
CONFIG_SCSI_SIM710=m
CONFIG_SCSI_SYM53C416=m
CONFIG_SCSI_DC390T=m
CONFIG_SCSI_T128=m
CONFIG_SCSI_U14_34F=m
CONFIG_SCSI_U14_34F_MAX_TAGS=8
CONFIG_SCSI_ULTRASTOR=m
CONFIG_SCSI_NSP32=m
```

```
CONFIG_SCSI_DEBUG=m
CONFIG_SCSI_PCMCIA=y
CONFIG_PCMCIA_AHA152X=m
CONFIG_PCMCIA_FDOMAIN=m
CONFIG_PCMCIA_NINJA_SCSI=m
CONFIG_PCMCIA_QLOGIC=m
CONFIG_FUSION=m
CONFIG_FUSION_MAX_SGE=40
CONFIG_FUSION_ISENSE=m
CONFIG_FUSION_CTL=m
CONFIG_FUSION_LAN=m
CONFIG_NET_FC=y
CONFIG_IEEE1394=m
CONFIG_IEEE1394_PCILYNX=m
CONFIG_IEEE1394_OHCI1394=m
CONFIG_IEEE1394_VIDEO1394=m
CONFIG_IEEE1394_SBP2=m
CONFIG_IEEE1394_ETH1394=m
CONFIG_IEEE1394_DV1394=m
CONFIG_IEEE1394_RAWIO=m
CONFIG_IEEE1394_CMP=m
CONFIG_IEEE1394_AMDTP=m
CONFIG_I2O=m
CONFIG_I2O_PCI=m
CONFIG_I2O_BLOCK=m
CONFIG_I2O_LAN=m
CONFIG_I2O_SCSI=m
CONFIG_I2O_PROC=m
CONFIG_NETDEVICES=y
CONFIG_ARCNET=m
CONFIG_ARCNET_1201=m
CONFIG_ARCNET_1051=m
CONFIG_ARCNET_RAW=m
CONFIG_ARCNET_COM90xx=m
CONFIG_ARCNET_COM90xxIO=m
CONFIG_ARCNET_RIM_I=m
CONFIG_ARCNET_COM20020=m
CONFIG_ARCNET_COM20020_ISA=m
CONFIG_ARCNET_COM20020_PCI=m
CONFIG_DUMMY=m
CONFIG_BONDING=m
CONFIG_EQUALIZER=m
CONFIG_TUN=m
CONFIG_ETHERTAP=m
CONFIG_NET_SB1000=m
CONFIG_NET_ETHERNET=y
CONFIG_HAPPYMEAL=m
CONFIG_SUNGEM=m
CONFIG_NET_VENDOR_3COM=y
CONFIG_EL1=m
CONFIG_EL2=m
CONFIG_ELPLUS=m
CONFIG_EL16=m
CONFIG_EL3=m
CONFIG_3C515=m
CONFIG_VORTEX=m
CONFIG_TYPHOON=m
CONFIG_LANCE=m
CONFIG_NET_VENDOR_SMC=y
CONFIG_WD80x3=m
CONFIG_ULTRA=m
CONFIG_SMC9194=m
CONFIG_NET_VENDOR_RACAL=y
CONFIG_NI5010=m
```

```
CONFIG_NI52=m
CONFIG_NI65=m
CONFIG_AT1700=m
CONFIG_DEPCA=m
CONFIG_HP100=m
CONFIG_NET_ISA=y
CONFIG_E2100=m
CONFIG_EWRK3=m
CONFIG_EEXPRESS=m
CONFIG_EEXPRESS_PRO=m
CONFIG_HPLAN_PLUS=m
CONFIG_HPLAN=m
CONFIG_LP486E=m
CONFIG_ETH16I=m
CONFIG_NE2000=m
CONFIG_NET_PCI=y
CONFIG_PCNET32=m
CONFIG_AMD8111_ETH=m
CONFIG_ADAPTEC_STARFIRE=m
CONFIG_AC3200=m
CONFIG_APRICOT=m
CONFIG_B44=m
CONFIG_CS89x0=m
CONFIG_TULIP=m
CONFIG_DE4X5=m
CONFIG_DGRS=m
CONFIG_DM9102=m
CONFIG_EEPRO100=m
CONFIG_E100=m
CONFIG_FEALNX=m
CONFIG_NATSEMI=m
CONFIG_NE2K_PCI=m
CONFIG_FORCEDETH=m
CONFIG_8139CP=m
CONFIG_8139TOO=m
CONFIG_8139TOO_TUNE_TWISTER=y
CONFIG_8139TOO_8129=y
CONFIG_SIS900=m
CONFIG_EPIC100=m
CONFIG_SUNDANCE=m
CONFIG_TLAN=m
CONFIG_VIA_RHINE=m
CONFIG_WINBOND_840=m
CONFIG_NET_POCKET=y
CONFIG_ATP=m
CONFIG_DE600=m
CONFIG_DE620=m
CONFIG_ACENIC=m
CONFIG_DL2K=m
CONFIG_E1000=m
CONFIG_NS83820=m
CONFIG_HAMACHI=m
CONFIG_YELLOWFIN=m
CONFIG_R8169=m
CONFIG_SK98LIN=m
CONFIG_TIGON3=m
CONFIG_FDDI=y
CONFIG_DEFXX=m
CONFIG_SKFP=m
CONFIG_PLIP=m
CONFIG_PPP=m
CONFIG_PPP_MULTILINK=y
CONFIG_PPP_FILTER=y
CONFIG_PPP_ASYNC=m
```

```
CONFIG_PPP_SYNC_TTY=m
CONFIG_PPP_DEFLATE=m
CONFIG_PPP_BSDCOMP=m
CONFIG_PPPOE=m
CONFIG_SLIP=m
CONFIG_SLIP_COMPRESSED=y
CONFIG_SLIP_SMART=y
CONFIG_NET_RADIO=y
CONFIG_STRIP=m
CONFIG_WAVELAN=m
CONFIG_ARLAN=m
CONFIG_AIRONET4500=m
CONFIG_AIRONET4500_NONCS=m
CONFIG_AIRONET4500_PNP=y
CONFIG_AIRONET4500_PCI=y
CONFIG_AIRONET4500_PROC=m
CONFIG_AIRO=m
CONFIG_HERMES=m
CONFIG_PLX_HERMES=m
CONFIG_TMD_HERMES=m
CONFIG_PCI_HERMES=m
CONFIG_PCMCIA_HERMES=m
CONFIG_AIRO_CS=m
CONFIG_PCMCIA_ATMEL=m
CONFIG_PRISM54=m
CONFIG_FW_LOADER=m
CONFIG_NET_WIRELESS=y
CONFIG_TR=y
CONFIG_IBMTR=m
CONFIG_IBMOL=m
CONFIG_IBMLS=m
CONFIG_3C359=m
CONFIG_TMS380TR=m
CONFIG_TMSPCI=m
CONFIG_TMSISA=m
CONFIG_ABYSS=m
CONFIG_SMCTR=m
CONFIG_NET_FC=y
CONFIG_IPHASE5526=m
CONFIG_RCPCI=m
CONFIG_SHAPER=m
CONFIG_WAN=y
CONFIG_HOSTESS_SV11=m
CONFIG_COSA=m
CONFIG_DSCC4=m
CONFIG_LANMEDIA=m
CONFIG_ATI_XX20=m
CONFIG_SEALEVEL_4021=m
CONFIG_SYNCLINK_SYNCPPP=m
CONFIG_HDLC=m
CONFIG_HDLC_RAW=y
CONFIG_HDLC_RAW_ETH=y
CONFIG_HDLC_CISCO=y
CONFIG_HDLC_FR=y
CONFIG_PCI200SYN=m
CONFIG_PC300=m
CONFIG_FARSYNC=m
CONFIG_N2=m
CONFIG_C101=m
CONFIG_DLCI=m
CONFIG_DLCI_COUNT=24
CONFIG_DLCI_MAX=8
CONFIG_SDLA=m
CONFIG_WAN_ROUTER_DRIVERS=y
```

```
CONFIG_VENDOR_SANGOMA=m
CONFIG_WANPIPE_CHDLC=y
CONFIG_WANPIPE_PPP=y
CONFIG_CYCLADES_SYNC=m
CONFIG_CYCLOMX_X25=y
CONFIG_LAPBETHER=m
CONFIG_X25_ASY=m
CONFIG_SBNI=m
CONFIG_NET_PCMCIA=y
CONFIG_PCMCIA_3C589=m
CONFIG_PCMCIA_3C574=m
CONFIG_PCMCIA_FMVJ18X=m
CONFIG_PCMCIA_PCNET=m
CONFIG_PCMCIA_AXNET=m
CONFIG_PCMCIA_NMCLAN=m
CONFIG_PCMCIA_SMC91C92=m
CONFIG_PCMCIA_XIRC2PS=m
CONFIG_ARCNET_COM20020_CS=m
CONFIG_PCMCIA_IBMTR=m
CONFIG_PCMCIA_XIRCOM=m
CONFIG_PCMCIA_XIRTULIP=m
CONFIG_NET_PCMCIA_RADIO=y
CONFIG_PCMCIA_RAYCS=m
CONFIG_PCMCIA_NETWAVE=m
CONFIG_PCMCIA_WAVELAN=m
```

Da mesma forma que algumas configurações foram retiradas do kernel, outras foram adicionadas, conforme mostra o quadro abaixo:

```
CONFIG_MK7=y
CONFIG_X86_L1_CACHE_SHIFT=6
CONFIG_X86_HAS_TSC=y
CONFIG_X86_GOOD_APIC=y
CONFIG_X86_USE_3DNOW=y
CONFIG_X86_PGE=y
CONFIG_X86_USE_PPRO_CHECKSUM=y
CONFIG_X86_F00F_WORKS_OK=y
CONFIG_X86_TSC=y
CONFIG_BINFMT_AOUT=y
CONFIG_BINFMT_MISC=y
```

# Anexo 2: Kernel do OpenBSD – Configuração

No caso do OpenBSD, a recompilação de kernel ocorre de uma forma um pouco diferente do que ocorre no Linux. Dessa forma, o arquivo **/usr/src/sys/arch/i386/conf/FABIO** foi criado a partir do /usr/src/src/arch/i386/conf/GENERIC. No quadro abaixo, estão as opções que foram retiradas desse arquivo para que o sistema fosse customizado.

```
include "../../../conf/GENERIC"
option          I386_CPU        # CPU classes; at least one is REQUIRED
option          I486_CPU
option          I586_CPU
option          GPL_MATH_EMULATE # floating point emulation.
option          COMPAT_SVR4# binary compatibility with SVR4
option          COMPAT_IBCS2        # binary compatibility with SCO and ISC
option          COMPAT_LINUX        # binary compatibility with Linux
option          COMPAT_FREEBSD      # binary compatibility with FreeBSD
option          COMPAT_BSDOS        # binary compatibility with BSD/OS
option          COMPAT_AOUT         # a.out binaries are emulated
elansc* at pci? dev ? function ?    # AMD Elan SC520 System Controller
gpio*   at elansc?
geodesc* at pci? dev ? function ?   # Geode SC1100/SCx200 IAOC
lm0     at isa? port 0x290
nsclpcsio* at isa?                  # NS PC87366 LPC Super I/O
gpio*   at nsclpcsio?
gscsio* at isa?                         # NS Geode SC1100 Super I/O
iic0    at gscsio?                      # ACCESS.bus 1
iic1    at gscsio?                      # ACCESS.bus 2
lmtemp0at iic1 addr 0x48            # NS LM75/LM77 temperature sensor
pcic0   at isa? port 0x3e0 iomem 0xd0000 iosiz 0x10000
pcic1   at isa? port 0x3e2 iomem 0xe0000 iosiz 0x4000
pcic2   at isa? port 0x3e4 iomem 0xe0000 iosiz 0x4000
tcic0   at isa? disable port 0x240 iomem 0xd0000 iosiz 0x10000
pcic*   at isapnp?
pcic*   at pci? dev? function ?
pcmcia* at pcic? controller ? socket ?
pcmcia* at tcic? controller ? socket ?
cardbus*        at cardslot?
pcmcia*         at cardslot?
cbb*            at pci? dev ? function ?
cardslot*       at cbb?
ehci*   at pci?                         # Enhanced Host Controller
uhci*   at pci?                         # Universal Host Controller (Intel)
ohci*   at pci?                         # Open Host Controller
usb*    at ehci?
usb*    at uhci?
usb*    at ohci?
uhub*   at usb?                         # USB Hubs
uhub*   at uhub? port ? configuration ?     # USB Hubs
umodem*         at uhub? port ? configuration ?     # USB Modems/Serial
ucom*   at umodem?
uvisor* at uhub? port ? configuration ?     # Handspring Visor
ucom*   at uvisor?
uvscom*         at uhub? port ?                 # SUNTAC Slipper U VS-10U serial
ucom*   at uvscom? portno ?
ubsa*   at uhub? port ?                 # Belkin serial adapter
ucom*   at ubsa? portno ?
uftdi*  at uhub? port ?                 # FTDI FT8U100AX serial adapter
ucom*   at uftdi? portno ?
uplcom* at uhub? port ?                 # I/O DATA USB-RSAQ2 serial adapter
```

```
ucom*   at uplcom? portno ?
umct*   at uhub? port ?                        # MCT USB-RS232 serial adapter
ucom*   at umct? portno ?
uaudio* at uhub? port ? configuration ?        # USB Audio
audio*  at uaudio?
umidi*  at uhub? port ? configuration ?        # USB MIDI
midi*   at umidi?
ulpt*   at uhub? port ? configuration ? # USB Printers
umass*  at uhub? port ? configuration ? # USB Mass Storage devices
scsibus* at umass?
uhidev* at uhub? port ? configuration ? interface ?     # Human Interface Devices
ums*    at uhidev? reportid ?                  # USB mouse
wsmouse* at ums? mux 0
ukbd*   at uhidev? reportid ?                  # USB keyboard
wskbd*  at ukbd? console ? mux 1
uhid*   at uhidev? reportid ?                  # USB generic HID support
aue*    at uhub? port ?                        # ADMtek AN986 Pegasus Ethernet
axe*    at uhub? port ?                        # ASIX Electronics AX88172 USB Ethernet
cue*    at uhub? port ?                        # CATC USB-EL1201A based Ethernet
kue*    at uhub? port ?                        # Kawasaki KL5KUSB101B based Ethernet
cdce*   at uhub? port ?                        # CDC Ethernet
upl*    at uhub? port ?                        # Prolific PL2301/PL2302 host-to-host `network'
url*    at uhub? port ?                        # Realtek RTL8150L based adapters
wi*     at uhub? port ?                        # WaveLAN IEEE 802.11DS
urio*   at uhub? port ?                        # Diamond Multimedia Rio 500
uscanner* at uhub? port ?              # USB Scanners
usscanner* at uhub? port ?             # USB SCSI scanners, e.g., HP5300
scsibus* at usscanner?
uyap*   at uhub? port ?                        # Y@P firmware loader
udsbr*  at uhub? port ?                        # D-Link DSB-R100 radio
radio*  at udsbr?                              # USB radio
ugen*   at uhub? port ? configuration ? # USB Generic driver
pccom*  at pcmcia? function ?                  # PCMCIA modems/serial ports
bha0    at isa? port 0x330 irq ? drq ?         # BusLogic [57]4X SCSI controllers
bha1    at isa? disable port 0x334 irq ? drq ? # BusLogic [57]4X SCSI controllers
bha2    at isa? disable port ? irq ?
bha*    at pci? dev ? function ?
scsibus* at bha?
aha0    at isa? port 0x330 irq ? drq ?         # Adaptec 154[02] SCSI controllers
aha1    at isa? port 0x334 irq ? drq ?         # Adaptec 154[02] SCSI controllers
aha*    at isapnp?
scsibus* at aha?
ahb*    at eisa? slot ?                        # Adaptec 174[024] SCSI controllers
scsibus* at ahb?
ahc0    at isa? port ? irq ?           # Adaptec 284x SCSI controllers
ahc*    at eisa? slot ?                        # Adaptec 274x, aic7770 SCSI controllers
ahc*    at pci? dev ? function ?       # Adaptec 2940/3940/78?? SCSI controllers
scsibus* at ahc?
ahd*    at pci? dev ? function ?       # Adaptec 79?? SCSI controllers
scsibus* at ahd?
mpt*    at pci? dev ? function ?       # LSI Fusion-MPT SCSI/Fibre
scsibus* at mpt?
dpt*    at pci? dev ? function ?       # DPT SmartCache/SmartRAID PCI
dpt*    at eisa? slot ?                        # DPT SmartCache/SmartRAID EISA
scsibus* at dpt?
gdt*    at pci? dev ? function ?       # ICP Vortex GDT RAID controllers
scsibus* at gdt?
twe*    at pci? dev ? function ?       # 3ware Escalade RAID controllers
scsibus* at twe?
aac*    at pci? dev ? function ?       # Adaptec FSA RAID controllers
scsibus* at aac?
ami*    at pci? dev ? function ?       # AMI MegaRAID controllers
scsibus* at ami?
cac*    at pci? dev ? function ?       # Compaq Smart ARRAY RAID controllers
```

```
cac*      at eisa? slot ?
scsibus* at cac?
iha*      at pci? dev ? function ?     # Initio Ultra/UltraWide SCSI controllers
scsibus* at iha?
isp*      at pci? dev ? function ?     # Qlogic ISP [12]0x0 SCSI/FibreChannel
scsibus* at isp?
aic0      at isa? port 0x340 irq 11    # Adaptec 152[02] SCSI controllers
aic*      at pcmcia? function ?                # PCMCIA based aic SCSI controllers
aic*      at isapnp?                           # isapnp configured aic SCSI controllers
scsibus* at aic?
siop*     at pci? dev ? function ?     # NCR 538XX SCSI controllers(new)
scsibus* at siop?
adv*      at pci? dev ? function ?     # AdvanSys 1200A/B and ULTRA SCSI
scsibus* at adv?
adw*      at pci? dev ? function ?     # AdvanSys ULTRA WIDE SCSI
scsibus* at adw?
pcscp*    at pci? dev ? function ?     # AMD 53c974 PCscsi-PCI SCSI
scsibus* at pcscp?
sea0      at isa? disable iomem 0xc8000 irq 5          # Seagate ST0[12] SCSI controllers
scsibus* at sea?
trm*      at pci? dev ? function ?     # Tekram DC-3x5U SCSI Controllers
scsibus* at trm?
uha0      at isa? port 0x330 irq ? drq ?       # UltraStor [13]4f SCSI controllers
uha1      at isa? disable port 0x334 irq ? drq ? # UltraStor [13]4f SCSI controllers
uha*      at eisa? slot ?                      # UltraStor 24f SCSI controllers
scsibus* at uha?
wds0      at isa? disable port 0x350 irq 15 drq 6      # WD7000 and TMC-7000 controllers
scsibus* at wds?
sd*       at scsibus? target ? lun ?   # SCSI disk drives
st*       at scsibus? target ? lun ?   # SCSI tape drives
cd*       at scsibus? target ? lun ?   # SCSI CD-ROM drives
ch*       at scsibus? target ? lun ?   # SCSI autochangers
ss*       at scsibus? target ? lun ?   # SCSI scanners
uk*       at scsibus? target ? lun ?   # unknown SCSI
wdc*      at pcmcia? function ?
we0       at isa? port 0x280 iomem 0xd0000 irq 9       # WD/SMC 80x3 ethernet
we1       at isa? port 0x300 iomem 0xcc000 irq 10      #
we*       at isapnp?
ec0       at isa? port 0x250 iomem 0xd8000 irq 9       # 3C503 ethernet
ne0       at isa? port 0x240 irq 9             # NE[12]000 ethernet
ne1       at isa? port 0x300 irq 10            # NE[12]000 ethernet
ne2       at isa? port 0x280 irq 9             # NE[12]000 ethernet
ne*       at isapnp?                           # NE[12]000 PnP ethernet
eg0       at isa? disable port 0x310 irq 5     # 3C505/Etherlink+ ethernet
el0       at isa? disable port 0x300 irq 9     # 3C501 ethernet
ep0       at isa? port ? irq ?                 # 3C509 ethernet
ep*       at isapnp?                           # 3C509 PnP ethernet
ep*       at isa? port ? irq ?                 # 3C509 ethernet
ef*       at isapnp?                           # 3C515 PnP ethernet
ie0       at isa? port 0x360 iomem 0xd0000 irq 7       # StarLAN and 3C507
lc0       at isa? port 0x200 irq ?             # DEC EtherWorks
lc1       at isa? port 0x280 irq ?             # DEC EtherWorks
le0       at isa? port 0x360 irq 15 drq 6              # IsoLan, NE2100, and DEPCA
ex0       at isa? port 0x320 irq 5             # Intel EtherExpress PRO/10
ep0       at eisa? slot ?
ep*       at eisa? slot ?                      # 3C579 ethernet
fea*      at eisa? slot ?                      # DEC DEFEA FDDI
lmc*      at pci? dev ? function ?             # Lan Media Corp SSI/T3/HSSI
san*      at pci? dev ? function ?             # Sangoma PCI AFT card
le*       at pci? dev ? function ?             # PCnet-PCI based ethernet
le*       at isapnp?
de*       at pci? dev ? function ?             # DC21X4X-based ethernet
fxp*      at pci? dev ? function ?             # EtherExpress 10/100B ethernet
fxp*      at cardbus? dev ? function ?                 # Intel PRO/100 ethernet
```

```
ne*     at pci? dev ? function ?        # NE2000-compat ethernet
ep0     at pci? dev ? function ?        # 3C59x ethernet
ep*     at pci? dev ? function ?        # 3C59x ethernet
ne*     at pcmcia? function ?                   # PCMCIA based NE2000 ethernet
ep*     at pcmcia? function ?                   # PCMCIA based 3C5xx ethernet
sm*     at pcmcia? function ?                   # PCMCIA based sm ethernet
xe*     at pcmcia? function ?                   # Xircom ethernet
fpa*    at pci? dev ? function ?        # DEC DEFPA FDDI
xl*     at pci? dev ? function ?        # 3C9xx ethernet
xl*     at cardbus? dev ? function ?            # 3C575/3C656 ethernet
rl*     at pci? dev ? function ?        # RealTek 81[23]9 ethernet
rl*     at cardbus? dev ? function ?            # RealTek 81[23]9 ethernet
mtd*    at pci? dev ? function ?        # Myson MTD800/803/891
tx*     at pci? dev ? function ?        # SMC 83C170 EPIC ethernet
tl*     at pci? dev ? function ?        # Compaq Thunderlan ethernet
wb*     at pci? dev ? function ?        # Winbond W89C840F ethernet
sf*     at pci? dev ? function ?        # Adaptec AIC-6915 ethernet
ste*    at pci? dev ? function ?        # Sundance ST201 ethernet
dc*     at pci? dev ? function ?        # 21143, "tulip" clone ethernet
dc*     at cardbus? dev ? function ?            # 21143, "tulip" clone ethernet
ti*     at pci? dev ? function ?        # Alteon Tigon 1Gb ethernet
skc*    at pci? dev ? function ?        # SysKonnect GEnesis 984x
sk*     at skc?                                 #  each port of above
em*     at pci? dev ? function ?        # Intel Pro/1000 ethernet
txp*    at pci? dev ? function ?        # 3com 3CR990
nge*    at pci? dev ? function ?        # NS DP83820/DP83821 GigE
bge*    at pci? dev ? function ?        # Broadcom BCM570x (aka Tigon3)
re*     at pci? dev ? function ?        # Realtek 8169/8169S/8110S
stge*   at pci? dev ? function ?        # Sundance TC9021 GigE
hme*    at pci? dev ? function ?        # Sun Happy Meal
bce*    at pci? dev ? function ?        # Broadcom BCM4401
atw*    at pci? dev ? function ?        # ADMtek ADM8211 (802.11)
atw*    at cardbus? dev ? function ?            # ADMtek ADM8211 (802.11)
wi*     at pci? dev ? function ?        # WaveLAN IEEE 802.11DS
wi*     at pcmcia? function ?                   # WaveLAN IEEE 802.11DS
an*     at pci? dev ? function ?        # Aironet IEEE 802.11DS
an*     at isapnp?                              # Aironet IEEE 802.11DS
an*     at pcmcia? function ?                   # Aironet IEEE 802.11DS
ray*    at pcmcia? function ?                   # Raylink Aviator2.4/Pro 802.11FH
exphy*  at mii? phy ?                   # 3Com internal PHYs
inphy*  at mii? phy ?                   # Intel 82555 PHYs
iophy*  at mii? phy ?                   # Intel 82553 PHYs
icsphy* at mii? phy ?                   # ICS 1890 PHYs
lxtphy* at mii? phy ?                   # Level1 LXT970 PHYs
nsphy*  at mii? phy ?                   # NS and compatible PHYs
nsphyter*       at mii? phy ?           # NS and compatible PHYs
qsphy*  at mii? phy ?                   # Quality Semi QS6612 PHYs
sqphy*  at mii? phy ?                   # Seeq 8x220 PHYs
rlphy* at mii? phy ?                    # RealTek 8139 internal PHYs
mtdphy* at mii? phy ?                   # Myson MTD972 PHYs
dcphy*  at mii? phy ?                   # Digital Clone PHYs
acphy*  at mii? phy ?                   # Altima AC101 PHYs
amphy*  at mii? phy ?                   # AMD 79C873 PHYs
tqphy*  at mii? phy ?                   # TDK 78Q212x PHYs
bmtphy* at mii? phy ?                   # Broadcom 10/100 PHYs
brgphy* at mii? phy ?                   # Broadcom Gigabit PHYs
eephy* at mii? phy ?                    # Marvell 88E1000 series PHY
xmphy* at mii? phy ?                    # XaQti XMAC-II PHYs
nsgphy* at mii? phy ?                   # NS gigabit PHYs
urlphy*  at mii? phy ?                  # Realtek RTL8150L internal PHY
rgephy*  at mii? phy ?                  # Realtek 8169S/8110S PHY
ukphy*  at mii? phy ?                   # "unknown" PHYs
pss0    at isa? port 0x220 irq 7 drq 6          # Personal Sound System
sp0     at pss0 port 0x530 irq 10 drq 0         # sound port driver
```

```
eap*    at pci? dev ? function ?                # Ensoniq AudioPCI S5016
eso*    at pci? dev ? function ?                # ESS Solo-1 PCI AudioDrive
sv*     at pci? dev ? function ?                # S3 SonicVibes (S3 617)
neo*    at pci? dev ? function ?                # NeoMagic 256AV/ZX
cmpci*  at pci? dev ? function ?                # C-Media CMI8338/8738
auich*  at pci? dev ? function ? flags 0x0000          # i82801 ICH AC'97 audio
autri*  at pci? dev ? function ? flags 0x0000          # Trident 4D WAVE
auvia*  at pci? dev ? function ?                # VIA VT82C686A
clcs*   at pci? dev ? function ?                # CS4280 CrystalClear audio
clct*   at pci? dev ? function ?                # CS4281 CrystalClear audio
fms*    at pci? dev ? function ?                # Forte Media FM801
maestro* at pci? dev ? function ?               # ESS Maestro PCI
esa*    at pci? dev ? function ?                # ESS Maestro3 PCI
yds*    at pci? dev ? function ? flags 0x0000          # Yamaha YMF Audio
emu*    at pci? dev ? function ?                # SB Live!
sb0     at isa? port 0x220 irq 5 drq 1                 # SoundBlaster
sb*     at isapnp?
ess*    at isapnp?                                     # ESS Tech ES188[78], ES888
wss0    at isa? port 0x530 irq 10 drq 0                # Windows Sound System
wss*    at isapnp?
pas0    at isa? port 0x220 irq 7 drq 1                 # ProAudio Spectrum
gus0    at isa? port 0x220 irq 7 drq 1 drq2 6          # Gravis (drq2 is record drq)
ym*     at isapnp?
mpu*    at isapnp?
mpu*    at isa? port 0x300                             # generic MPU, Yamaha SW60XG
opl*    at eso?
opl*    at sb?
opl*    at ess?
opl*    at yds?
midi*   at pcppi?               # MIDI interface to the PC speaker
midi*   at sb?                  # SB MPU401 port
midi*   at opl?                 # OPL FM synth
midi*   at ym?
midi*   at mpu?
midi*   at autri?
audio*  at sb?
audio*  at gus?
audio*  at pas?
audio*  at sp?
audio*  at ess?
audio*  at wss?
audio*  at ym?
audio*  at eap?
audio*  at eso?
audio*  at sv?
audio*  at neo?
audio*  at cmpci?
audio*  at clcs?
audio*  at clct?
audio*  at auich?
audio*  at autri?
audio*  at auvia?
audio*  at fms?
audio*  at maestro?
audio*  at esa?
audio*  at yds?
audio*  at emu?
bktr0   at pci? dev ? function ?
radio*  at bktr?
radio*  at fms?
joy*    at isapnp?
hifn*   at pci? dev ? function ?    # Hi/fn 7751 crypto card
lofn*   at pci? dev ? function ?    # Hi/fn 6500 crypto card
nofn*   at pci? dev ? function ?    # Hi/fn 7814/7851/7854 crypto card
```

```
ubsec*  at pci? dev ? function ?    # Bluesteel Networks 5xxx crypto card
safe*   at pci? dev ? function ?    # SafeNet SafeXcel 1141/1741
iop*    at pci? dev ? function ?    # I2O
ioprbs* at iop? tid ?                       # Random block storage
scsibus* at ioprbs?
pseudo-device   pctr            1
pseudo-device   mtrr            1       # Memory range attributes control
pseudo-device   sequencer       1
pseudo-device   bio             1       # ioctl multiplexing device
pseudo-device   hotplug         1       # devices hot plugging
pseudo-device   wsmux           2
pseudo-device   crypto          1
```

Da mesma forma, outra linha foi adicionada para que se incluísse o arquivo /usr/src/sys/conf/FABIO, que foi copiado do arquivo /usr/src/sys/conf/GENERIC. A opção abaixo foi adicionada ao arquivo /usr/src/sys/arch/i386/conf/FABIO:

```
include "../../../conf/FABIO"
```

O arquivo /**usr/src/sys/conf/FABIO**, que foi adicionado na linha acima ao arquivo /usr/src/sys/arch/i386/conf/FABIO, teve as seguintes opções removidas:

```
option          DDB             # in-kernel debugger
option          CRYPTO                  # Cryptographic framework
option          UVM_SWAP_ENCRYPT# support encryption of pages going to swap
option          QUOTA           # UFS quotas
option          EXT2FS          # Second Extended Filesystem
option          MFS             # memory file system
option          XFS             # xfs filesystem
option          NFSCLIENT       # Network File System client
option          NFSSERVER       # Network File System server
option          CD9660          # ISO 9660 + Rock Ridge file system
option          MSDOSFS                 # MS-DOS file system
option          FDESC           # /dev/fd
option          INET6           # IPv6 (needs INET)
option          IPSEC           # IPsec
option          PPP_BSDCOMP # PPP BSD compression
option          PPP_DEFLATE
pseudo-device   sppp    1       # Sync PPP/HDLC
pseudo-device   enc     1       # option IPSEC needs the encapsulation interface
pseudo-device   bridge          # network bridging support
pseudo-device   carp            # CARP protocol support
pseudo-device   gif             # IPv[46] over IPv[46] tunnel (RFC1933)
pseudo-device   gre             # GRE encapsulation interface
pseudo-device   ppp             # PPP
pseudo-device   sl              # CSLIP
pseudo-device   tun             # network tunneling over tty
pseudo-device   vlan            # IEEE 802.1Q VLAN
```