José Geraldo de Oliveira

QoS transparente com GNU/Linux

Monografia de Pós-graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux".

Orientador
Prof. Denilson V. Martins

Lavras
Minas Gerais - Brasil
2009

José Geraldo de Oliveira

QoS transparente com GNU/Linux

Monografia de Pós-graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux".

Aprovada em *Novembro de 2009*

Prof. Sandro Melo

Prof. Arlindo Follador Neto

Prof. Denilson V. Martins
(Orientador)

Lavras
Minas Gerais - Brasil
2009

Para minha esposa Maria Helena e filhos Débora Elisa e Paulo Fernando, pelo carinho e apoio que me deram durante a realização deste trabalho.

Agradecimentos

Agradeço a Deus, a minha família maravilhosa e a todos que colaboraram direta e indiretamente com este trabalho. Agradeço particularmente aos meus pais, que me mostraram desde cedo a importância da perseverança e da ética, entre outros valores.

Sumário

Lista de Siglas

AMD *Advanced Micro Devices* - nome de uma grande fabricante mundial de circuitos integrados, especialmente processadores. 35

ATM *Asynchronous Transfer Mode* - modo de transferência assíncrona. 5

CPD Centro de Processamento de Dados. 18

DMZ *Demilitarized zone* - zona desmilitarizada. 17

ERP *Enterprise Resource Planning* - planejamento de recursos da empresa. 14, 16

FIFO First In, First Out - primeiro a entrar, primeiro a sair. 7, 10

FTP *File Transfer Protocol* - protocolo de transferência de arquivo. 11, 38, 39, 41, 43

FTP-DATA *File Transfer Protocol Data* - protocolo de transferência de arquivo, subconjunto de tráfego de dados. 38, 39, 41

G.711 Padronização, pela ITU-T, de uma técnica de codificação para voz em PCM a taxa de 64 Kbps. 6

H.261 Padronização, pela ITU-T, de uma técnica de codificação de imagens móveis a taxas de 64 a 1920 Kbps. 6

HP *Hewlett Packard* - nome de uma grande fabricante mundial de computadores. 35

HTB *Hierarchical Token Bucket* - balde hierárquico de fichas. ix, 3, 13, 17–20, 22–26, 28, 30, 35, 37, 39, 43, 44

HTTP Hypertext Transfer Protocol - protocolo de transferência de hipertexto. 38, 39, 41, 43, 44

HTTPS Hypertext Transfer Protocol Secure - protocolo seguro de transferência de hipertexto. 38, 43

IP *Internet Protocol* - Protocolo de Internet. ix, 10, 26, 62, 63

IPv4 *Internet Procotol version 4* - protocolo de Internet versão 4. 2, 7

IPv6 *Internet Procotol version 6* - protocolo de Internet versão 6. 2

ITU-T ITU Telecommunication Standardization Sector - Organização responsável por definir e coordenar padronizações, geralmente não obrigatórias, relacionadas a telecomunicações. iii

Kbps *Kilo bit per second* - quilo bit por segundo. iii

LAN *Local Area Network* - rede local. 8, 16

MMORPG *Massive Multiplayer Online Role Playing Game* - jogo online para multiplos jogadores que interpretam o papel do jogador. 1

MPLS *Multiprotocol Label Switching* - chaveamento de rótulo multi protocolo. 5, 17

OSPF *Open Shortest Path First* - abra primeiro o menor caminho. 14

PCM Pulse-code Modulation - modulação por código de pulsos é uma técnica de amostragem de sinal, permitindo transformar sinal analógico em digital. iii

QoE *Quality of Experience* - qualidade da experiência (do usuário). ix, 43

QoS *Quality of Service* - qualidade do serviço. ix, 2, 3, 5, 7–9, 17, 18, 22, 23, 43, 44

RAID-1 *Redundand Array of Inexpensive Disks* - conjunto redundante de discos econômicos. 14, 19

RAM *Random Access Memory* - memória de acesso aleatório. 19

SCP Secure Copy - cópia segura. 43

SCSI *Small Computer System Interface* - interface de sistema para pequenos computadores. 19

SFTP Secure File Transfer Protocol - protocolo de transferência segura de arquivos. 43

SSH Secure Shell - shell seguro. SSH é, ao mesmo tempo, um protocolo seguro e um programa para emulação de terminal. 38, 43

T3 é uma notação comum em telecomunicações e indica uma conexão digital de dados em velocidades que variam de 3Mbps a 45Mbps. 7

TCP *Transmit Control Protocol* - protocolo de controle de transmissão. 8, 10, 36, 62

TCP/IP *Transmit Control Protocol / Internet Protocol* - protocolo de controle de transmissão / protocolo de Internet. 7, 36

ToS *Type of Service* - tipo de serviço. 10

UDP *User Datagram Protocol* - protocolo de datagramas de usuário. 8, 10, 16

UTP *Unshielded Twisted Pair* - par trançado sem blindagem. 6

VoIP *Voice over IP* - voz sobre o protocolo IP. ix, 1–3, 11, 15

WAN *Wide Area Network* - rede de longa distância. 3, 8, 14–17

Lista de Figuras

Resumo

As redes com protocolo IP conectam milhões de dispositivos ao redor do mundo, principalmente através da Internet. Como este protocolo não possui suporte nativo a controle de qualidade de serviço (QoS), a experiência do usuário (QoE), principalmente com aplicações que demandam fluxo relativamente constante de dados, tais como fluxo de vídeo ou voz sobre IP (VoIP), é muitas vezes frustrante.

Este trabalho apresenta uma forma direta e prática para implantação de controle de qualidade de serviço (QoS), utilizando o recurso HTB do kernel do GNU/Linux operando em modo transparente, permitindo reserva, compartinhamento de largura de banda e/ou priorização de tráfego, com o objetivo de melhorar a experiência do usuário QoE.

Palavras-Chave: QoS; QoE; Linux; HTB; *Quality of Service*, *Quality of Experience*

Capítulo 1

Introdução

O crescimento constante da largura de banda de rede na última milha[1] para o consumidor, tem sido um grande motor para o crescimento dos *backbones* das operadoras de telecomunicações e também dos links de fornecedores de informação. Esta largura de banda de última milha, hoje já em torno dos 10Mbps[2] permite inúmeras implementações para melhorar a experiência do usuário, quando em navegação na Internet, como a inclusão de áudio e vídeo nas páginas web e nas aplicações, bem como páginas web muito dinâmicas. Os jogos *on line* do tipo MMORPG também têm crescido muito em utilização, características gráficas e, é claro, ocupação de banda de Internet. As redes 3G de celular apareceram mais recentemente como mais um canal de distribuição de conteúdo e contam com o lançamento de aparelhos móveis com características suficientes para utilização destas novas experiências, tais como o *Apple Iphone*, o *Google Android G1* ou o *HTC Touch Diamond*. A proliferação dos pontos de acesso sem fio à rede mundial, sejam implantados por empresas, usuários domésticos e governos, também se apresentam como fatores de alavancagem de demanda por largura de banda de rede.

O aumento do alcance da Internet e também da banda máxima disponibilizada para o usuário, permite a utilização da Internet por aplicações antigas e adaptadas a este novo meio, tais como de câmeras de segurança, videoconferência e transmissão de TV. Este aumento tem também alavancado soluções que surgiram depois da Internet, tais como telefonia VoIP, jogos online e redes sociais virtuais. As aplica-

[1] Termo derivado da expressão em inglês "last mile", que define a conexão desde o prédio da operadora de telecomunicações e a instalação (ou residência) do cliente

[2] 10 mega bits por segundo

ções mais tradicionais, principalmente o correio eletrônico, continuam ocupando seu lugar no tráfego da Internet.

Este trabalho apresenta um modelo de solução que visa melhorar ou otimizar a utilização dos links de dados, sejam eles de Internet ou privativos, por empresas e até por usuários domésticos. Esta melhoria de utilização reflete diretamente na melhoria da experiência do usuário quando utilizando serviços que passam por estes links.

1.1 Justificativa

A implementação de soluções de controle de banda buscam alguma ordem no tráfego demandado por todas as alternativas citadas anteriormente, já que o protocolo IPv4, sozinho, não possui implementação nativa de QoS. O IPv6 foi projetado com funcionalidades de QoS, mas a sua implementação na Internet e nas empresas, ainda não ocorreu em larga escala. Na empresa que o autor trabalha não existe nem estudo para adequação dos equipamentos para IPv6.

Várias empresas querem também utilizar a Internet para finalidades profissionais, sem dispensar a praticidade de aplicações de mensagens instantâneas e o baixo custo de soluções VoIP. Porém, estas mpresas também buscam ferramentas que permitam a utilização dos recursos de Internet disponibilizados aos seus funcionários com responsabilidade, garantindo que os links de acesso à Internet sejam utilizados principalmente para os fins que a empresa os destina, e não para utilização pelos funcionários para fins particulares, sem controle.

1.2 Objetivos

O objetivo deste trabalho é apresentar uma solução de Software Livre e Código Aberto (SLCA) para a implementação de uma solução de QoS, utilizando GNU/-Linux. Será demonstrado como foi feita uma implementação em uma grande empresa nacional.

A utilização de um modelo transparente (*bridge*) na rede ethernet apresenta um objetivo adicional, que é a rapidez que o servidor de QoS pode ser instalado e desinstalado fisicamente na rede, pois não demanda nenhuma alteração de rotas, faixas de rede e similares.

Apesar da *bridge* incluir uma camada adicional de processamento, aumentando consequentemente a latência, na empresa não é utilizada solução de vídeo-conferência nem de VoIP, de maneira que a latência acrescentada pela *bridge* não é perceptível.

1.3 Motivação e escopo

A motivação para este trabalho veio de um caso real. Na empresa para a qual o autor trabalha, era recorrente a ocorrência de reclamações de clientes internos e externos, sobre lentidões na rede e nas aplicações.

Análises com ferramentas de monitoração de rede, tais como **ntop**[3], **NetFlow Analyzer**[4] e **iptraf**[5], mostraram que realmente haviam alguns problemas de excesso de tráfego, algumas vezes provocados pela utilização de transferências de arquivos entre sites, compartilhamento de diretórios e *downloads* da Internet.

A empresa havia utilizado a solução de QoS da empresa *Blue Coat* anteriormente, e já estudado algumas soluções de otimização de WAN disponíveis no mercado. Considerando os custos envolvidos para adoção de uma solução proprietária tipo *Packet Shapper* e a estabilidade demonstrada pelo GNU/Linux dentro da empresa em outras atividades, a equipe de TI recomendou a implantação de uma solução de QoS com HTB, inicialmente no site de São Paulo, capital.

Este trabalho não se propõe a esgotar o assunto QoS em GNU/Linux. Também não detalha outras disciplinas de filas disponíveis em GNU/Linux, apesar de citá-las.

Não foi incluído neste trabalho outras opções de QoS existentes como software livre para outras plataformas tambem livres, tais como Dummynet para FreeBSD, ALTQ para FreeBSD e NetBSD,

Estas definições de limitação de escopo citadas aqui, decorreram da natureza do trabalho desenvolvido na empresa e da experiência com Red Hat Linux pela equipe interna de suporte. A pressão dos clientes internos pela resolução rápida para os problemas que se apresentavam, bem como a definição de uma solução de menor custo para o novo projeto alavancaram a utilização do Linux com HTB,

[3] http://www.ntop.org/overview.html
[4] http://www.manageengine.com/products/netflow/
[5] http://iptraf.seul.org/

uma solução que tinha sido citada em uma palestra interna por um fornecedor de soluções Linux.

1.4 Organização do Trabalho

Este trabalho está organizado da seguinte maneira: No capítulo 2 será apresentada uma fundamentação teórica de QoS e, a seguir, os tipos de QoS que estão disponíveis para serem implementados com GNU/Linux.

No capítulo 3 será apresentada uma explanação mais aprofundada do QoS com HTB, utilizando como exemplo a instalação e configuração efetuada na empresa que o autor trabalha.

No capítulo 4 serão apresentados os testes realizados e os resultados obtidos em laboratório.

No capítulo 5 será apresentada a conclusão do trabalho. Na seção seguinte, será apresentada a bibliografia utilizada e a seguir, os anexos.

Capítulo 2

Fundamentação Teórica sobre QoS

Esta seção tem como objetivo apresentar uma fundamentação teórica sobre QoS. Várias tipos de redes mais especializadas, entre elas a ATM e MPLS implementam nativamente QoS. Segundo (LEAL, 2004), QoS é conseguido através de negociações de vários parâmetros entre os componentes ativos da rede, com o objetivo de conceder a uma aplicação específica uma melhor qualidade no serviço prestado ao usuário.

Ainda segundo (LEAL, 2004), a aplicação deve especificar a faixa aceitável de valores para os parâmetros de QoS no momento da solicitação da conexão. Os valores solicitados são negociados fim a fim e, se os componentes envolvidos conseguirem atender a solicitação dentro dos limites definidos, a conexão será estabelecida. Entre os parâmetros para QoS negociaveis, os mais importantes são:

1. Retardo no estabelecimento da conexão
2. Vazão (*Throughput*)
3. Proteção
4. Prioridade

O "retardo no estabelecimento da conexão", como o próprio nome indica, é o tempo transcorrido entre a solicitação de conexão e o recebimento da confirmação. Quanto menor o valor deste parâmetro, melhor será o serviço.

A "vazão" é a da quantidade de bytes que serão trafegados em um intervalo de tempo, em cada direção. Cada aplicação inicialmente deve prever sua necessidade de fluxo de dados, solicitando a "reserva" no momento da conexão.

"Proteção" oferece uma forma da aplicação solicitar à camada de transporte, proteção contra a leitura ou modificação de dados por parte de terceiros.

O parâmetro "Prioridade" oferece uma maneira de garantir que as conexões de maior prioridade sejam atendidas com antecedência, em caso de congestionamento da rede ou partes da mesma.

É importante destacar ainda alguns parâmetros relacionados com a necessidade de cada aplicação. Estes parâmetros são a ***latência***, o ***jitter*** e o ***skew***.

Latência é o tempo que o dado demora na rede, sendo transportado de fim a fim. A *latência* pode ser provocada por:

- **Atraso na transmissão**. Este é o tempo decorrido entre o dado ter sido enviado por uma placa de rede até chegar na placa de rede do computador de destino. Contribui para este tempo a propagação do dado no meio físico (fibra ótica, cabo UTP e outros), o tempo de processamento em equipamentos intermediários (roteadores, *switches* e similares), fila de espera nestes equipamentos e efeitos correlatos.
- **Atraso na codificação e decodificação**. Dados como voz e vídeo precisam ser codificados antes de serem enviados para a rede. Este tempo depende do protocolo utilizado (G.711, H.261 e outros) e, é claro, da velocidade do processador da máquina.
- **Atraso de empacotamento e desempacotamento**. Depois que o dado está codificado, ele precisa ser "preparado" para ser enviado para a placa de rede. É necessário separar os dados, criar *frames* e encapsular, encaminhando então para a camada mais inferior, até chegar à placa de rede.

Jitter é a diferença de tempo entre a entrega dos pacotes. Algo como "flutuação da latência". Por causa do *jitter*, as aplicações precisam criar áreas de recepção chamadas *buffer* antes de iniciar a apresentação dos dados. Quanto maior o *jitter*, maior será o *buffer* necessário.

Skew é a diferença entre fluxos de dados separados, mas que se destinam a uma mesma experiência para o usuário. Em uma aplicação de videoconferência, por exemplo, o *skew* será a diferença entre a chegada do áudio e a chegada do vídeo, se os dois forem enviados por fluxos distintos. Neste exemplo ainda é

necessário considerar que o som propaga mais lentamente que o vídeo, tanto na captura quanto na entrega, fatores físicos que corroboram para o aparecimento do *skew* e uma piora da experiência do usuário.

2.1 QoS para IPv4

Conforme citado por (LEAL, 2004), o protocolo IPv4 não possui implementação nativa de QoS. Implementa apenas o algoritmo de "melhor tentativa", que é quase o mesmo que o algoritmo FIFO. Assim sendo, os parâmetros teoricamente possíveis para QoS em redes, tais como prioridade, retardo, taxa de erro e vazão não foram implementados no IPv4.

Devido à ampla utilização do IPv4 e da lacuna do protocolo com relação a QoS, surgiram algumas implementações que fornecem uma solução artificial de QoS. Estas soluções buscam atingir o objetivo de administrar o tráfego sem contudo alterar os princípios do protocolo IPv4. Tampouco incluem pacotes para negociação de parâmetros de QoS. Estas soluções de QoS implementam, simplificadamente, uma administração do tráfego baseado nos parâmetros configurados pelo administrador de sistemas. O administrador deve conhecer o tráfego que passa no link e criar as filas, prioridades e reservas desejadas para melhorar a utilização do recurso.

A administração do envio dos pacotes pelo QoS se beneficia da característica do TCP/IP em se adaptar à banda de comunicação disponível e até da capacidade de processamento dos equipamentos envolvidos. Normalmente pode ser visto na Internet uma estação conectada por um meio mais lento (digamos ADSL de 1Mbps) acessando um servidor conectado a uma velocidade bem mais alta, digamos uma conexão T3 a 45Mbps. Quando a estação requisita um objeto (uma figura, por exemplo), ambos negociam um parâmetro adaptável chamado "janela" (*window size*) [1].

O parâmetro *window size* informa quantos pacotes o computador enviará ao receptor, antes de receber um pacote de confirmação de recepção (*acknowledge*). Este pacote de confirmação informa ao transmissor que o pacote foi recebido corretamente, salvo no buffer de recepção e está disponível para a camada superior. Esta confirmação foi projetada para garantia de integridade de cada *frame* mas

[1] http://www.tcpipguide.com/free/t_TCPWindowSizeAdjustmentandFlowControl.htm, acessado em 29/06/2009

também é um tipo de QoS pois permite a adaptação do fluxo de dados aos links envolvidos na comunicação.

Baseado na taxa de recepção dos pacotes de confirmação e no tamanho da janela, o equipamento emissor vai ajustando a velocidade de envio, até conseguir um fluxo de dados relativamente constante.

2.2 Soluções de hardware para QoS em IPv4

Existem no mercado soluções denominadas "appliances", com sistema operacional e solução de QoS normalmente desconhecidas e proprietárias, mas que fazem o trabalho de maneira mais simplificada, do ponto de vista do administrador. Os produtos mais conhecidos deste grupo são:

- *Packet Shapper* [2] Desenvolvido pela **Packeteer**, hoje uma empresa da **Blue Coat**, é uma solução de rede LAN que atua de forma transparente na rede, coletando inicialmente informações do tráfego TCP, UDP e etc para depois criar um mapa de uso baseado em melhores práticas. O administrador pode alterar a proposta, mudando reservas de banda, priorizações e etc.

- *Wan Acceleration*, uma tecnologia implementada pela **Peribit Networks**, depois adquirida pela **Juniper Networks**[3]. Esta tecnologia de aceleração de WAN, simplificadamente, se parece com *cache* feito por soluções de *proxy*. Os roteadores da **Juniper Networks** implementam priorização, cache de objetos e "*local handshake*" entre outras oportunidades de diminuição de tráfego WAN. O "*local handshake*" responde aos pacotes de dados com *acknoledge* localmente, agindo como se a resposta originasse do servidor remoto, minimizando o tráfego e diminuindo a latência.

Estas duas soluções possuem uma ótima estabilidade pois não possuem peças móveis e os seus dados, tanto o sistema operacional quanto as configurações de regras, são gravadas em memória *flash*. Normalmente possuem ainda fontes de alimentação e interfaces de rede redundantes.

[2] Mais informações sobre *Packet Shapper* podem ser encontradas em http://www.bluecoat.com/products/packetshaper/

[3] Mais informações sobre a **Juniper Networks** e seus produtos podem ser encontradas em http://www.juniper.net

O “Packet Shapper” pode ser instalado em qualquer um dos lados do tráfego ou seja do lado do cliente ou do lado do servidor. É recomendado que seja do lado em que está o servidor. Já os roteadores com *Wan Acceleration* da **Juniper Networks** devem ser instalados em ambos os dados do tráfego.

2.3 QoS em IPv4 com GNU/Linux

O kernel do Linux suporta vários tipos de QoS, chamados de disciplina de fila ou resumidamente (como grande parte da a literatura os trata) *qdisc*. Cada umas destas disciplinas de filas é desenvolvida de um grupo diferente de pessoas e aderem ao kernel do Linux utilizando um acesso definido para tal fim. Estas camadas de software substituem a gerência de fila nativa do kernel.

Por característica, o GNU/Linux consegue efetuar o controle do fluxo somente na saída dos pacotes. Sendo o servidor de QoS um *firewall* ou *bridge*, o controle pode ser implantado nos dois sentidos pois o tráfego que “entra” por uma placa sai pela outra. Se o servidor for dedicado a outra atividade, um *web server* por exemplo, só será possível controlar o fluxo de dados de resposta das requisições recebidas.

2.3.1 Tipos de QoS com GNU/Linux

De acordo com (BROWN, 2006), o *Linux Traffic Control* implementa seis tipos de *qdiscs* sem classe, chamadas de *classless* e quatro com classes, chamadas *classfull*. A diferença entre as *qdiscs classless* e as *classfull* é que estas últimas suportam subdivisões, enquanto as *classless* não suportam. É oportuno citar que cada uma destas *qdiscs* foram implementadas por desenvolvedores diferentes. Desta maneira a documentação sobre algumas costuma ser bem fraca. Este trabalho não explora os detalhes teóricos de cada *qdisc*. Apenas as enumera, citando um resumo sobre cada uma. As *qdiscs classless* são:

- ***FIFO, First-In First-Out (pfifo and bfifo)***. Este algoritmo é a base para a fila padrão de todas as interfaces de rede, a fila *pfifo_fast*. Este algoritmo simplesmente transmite os pacotes à medida que eles chegam. Este também é o algoritmo padrão para uma nova classe, até que uma novo algoritmo seja definido. A diferença entre *pfifo* e *bfifo* é que em *pfifo* a definição do tamanho do *buffer* é feita por pacotes, enquanto que para *bfifo* é feita em bytes. Um modelo gráfico é mostrado na figura 2.1.

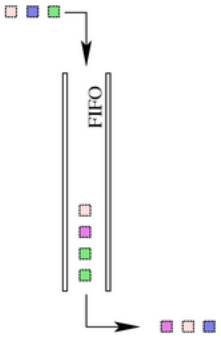

Figura 2.1: Disciplina de fila *First-in First-out* (FIFO)

- ***pfifo_fast***. Este algoritmo é o padrão para as interfaces em Linux (figura 2.2). Esta classe implementa três filas FIFO paralelas e com prioridades diferentes. Elas não podem ser alteradas pelo usuário. As prioridades são definidas pelos quatro bits ToS presentes no cabeçalho do datagrama IP, conforme (HUBERT, 2004). As dezesseis possibilidades dos bits de ToS são mapeadas pelo kernel para as três filas desta *qdisc*. A aderência de cada aplicação (telnet, smtp e etc) em cada uma das dezesseis possibilidades foi definida na RFC 1349, conforme (ALMQUIST, 1992).

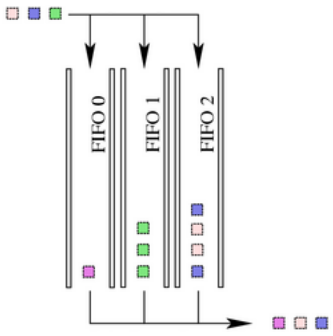

Figura 2.2: Disciplina de fila *pfifo_fast*

- ***SFQ, Stochastic Fair Queuing***. Esta *qdisc* utiliza um algoritmo que é uma implementação simplificada da família de algoritmos *fair queueing*. Esta *qdisc* é menos precisa que outras. A chave deste algoritmo é respeitar os fluxos de dados das sessões TCP ou fluxos UDP. Esta *qdisc* tem como objetivo impedir que o tráfego de uma sessão se sobreponha ao tráfego de outras sessões. Desta forma a *qdisc* simula a implementação de uma fila FIFO para

cada sessão. Veja um fluxo simulado na figura 2.3. A implementação das filas é feita através de algoritmo de *hash*.

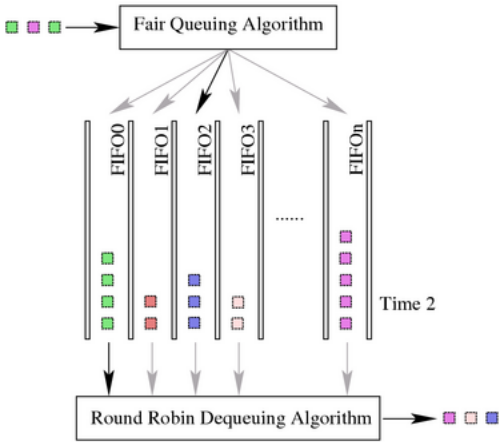

Figura 2.3: Disciplina de fila *Stochastic Fair Queuing*

- ***ESFQ, Extended Stochastic Fair Queuing***. Esta disciplina de fila é muito similar à *SFQ*. A diferença é que nesta implementação, o usuário pode escolher qual dos três algoritmos de HASH[4]. disponíveis será usado, entre *classic*, *source* ou *destination*.

- ***GRED, Generic Random Early Drop***. De acordo com (BALLIACHE, 2003), o GRED é um algoritmo com foco em definir a probabilidade de perda em uma rede congestionada para um tipo de tráfego. Pode-se assim definir, por exemplo, que não se quer perda em tráfego VoIP mas que aceita perdas de até 4% para tráfego FTP. A figura 2.4 demonstra a classe GRED.

- ***TBF, Token Bucket Filter***. Esta *qdisc* implementa um controle preciso de tráfego por fila. A característica marcante é que ela consegue garantir um excesso de tráfego durante um pequeno tempo.

As disciplinas de fila *classfull* são:

- ***PRIO, priority scheduler***. Esta fila funciona de uma maneira bem simples. Quando está pronta para enviar um pacote, a primeira classe é verificada. Se

[4]HASH é um tipo de algoritmo de dispersão que gera uma sequência de bits baseado no dado

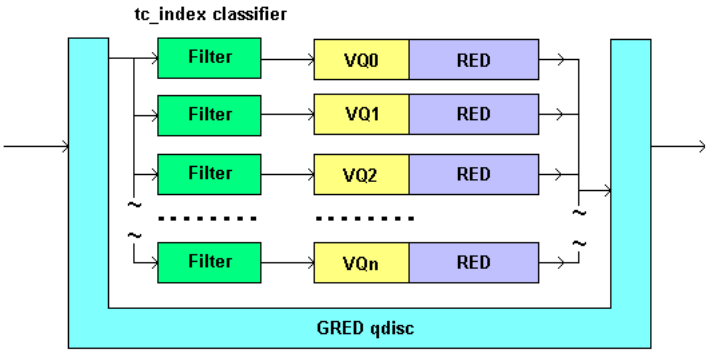

Figura 2.4: Disciplina de fila *Generic Random Early Drop*

tem um pacote, o mesmo é enviado. Se não tem, a próxima classe é verificada. Este processo continua até não haver mais classes a serem checadas.

- ***HFSC, Hierarchical Fair Service Curve***. De acordo com (RECHERT, 2005), HFSC usa um algoritmo hierárquico para gerenciar o tráfego, característica desejável em cenários complexos. HSFC permite, além do controle da distribuição da banda, o controle e alocação da latência.

- ***CBQ, Class Based Queuing***. Esta *qdisc* faz cálculos constantemente, para despachar pacotes em uma velocidade especifica, de maneira a enviar para o link a quantidade exata de dados que foi parametrizada. Para estes cálculos o CBQ utiliza "iddle timer" como unidade.

- ***HTB, Hierarchical Token Bucket***. Esta *qdisc* foi escrita depois da *CBQ*. De fato, ela funciona de maneira bem similar àquela mas, ao invés de usar "iddle time" para calcular a modelagem necessária para o tráfego, HTB usa o conceito de "token bucket" ou balde de fichas.

Capítulo 3

Implantação de QoS com HTB

A instalação de HTB descrita neste capítulo foi efetuada em uma grande empresa prestadora de serviços para os segmentos financeiro, de centrais de atendimento e de cobrança. O número de funcionários desta empresa em junho de 2.009, era por volta de oito mil. Ao final do segundo trimestre de 2.009 a empresa operava quase 3.600 (três mil e seiscentas) posições de atendimento em centrais de atendimento.

Esta empresa[1], para a qual este autor trabalha, possui sete escritórios no Brasil, nas seguintes cidades: Recife, Rio de Janeiro, Curitiba, São Paulo, Belo Horizonte e em Baruerí (condomínio Alphaville), estado de São Paulo. As filiais são interligadas por links dedicados *clear channel* de várias operadoras, formando duas estrelas com centros em São Paulo e em Belo Horizonte onde ficam, respectivamente, os centros comercial/financeiro e de tecnologia. Atualmente todos os escritórios possuem pontos de acesso para Internet, seja através de links dedicados ou através de conexões de band larga. A utilizão dos links demonstrados na figura 3.1 para acesso à Internet somente ocorre em contingência, quando a conexão de Internet do escritório fica paradada.

Em Belo Horizonte/MG e em São Paulo/SP ficam os servidores principais com os serviços de rede acessados por usuários de outros sites, tais como servidores *web*, servidores de correio eletrônico, servidores de terminal (*Microsoft Terminal Services*) e Mainframes. A maioria dos serviços são utilizados vinte e quatro horas por dia e sete dias na semana. A disponibilidade dos serviços é garantida pela utilização de equipamentos com características de alta disponibilidade. São utilizados *storages* com caminhos (no mínimo) duplicados até os servidores, discos

[1] A divulgação do nome da empresa não foi autorizada

internos de servidores em RAID-1, fontes de alimentação redundantes, *no breaks* e grupos geradores. Alguns servidores operam em *cluster* e outros possuem solução de contingência em outro prédio da própria companhia. A disponibilidade dos acessos aos serviços destes dois sites é melhorada pela utilização de links *clear channel* e do protocolo OSPF nos roteadores. A empresa usa vários servidores GNU/Linux para hospedar Bancos de Dados Oracle e servidores de aplicação *Red Hat JBOSS*. Utiliza também os softwares *open source* nagios, ntop, cacti, ocomon e xen, entre outros.

A topologia geral da rede WAN é demonstrada na figura 3.1. No *datacenter* da Totvs em São Paulo/SP, a empresa hospeda o seu ERP *Protheus* v10.0, desenvolvido pela Microsiga, hoje uma empresa do grupo Totvs. Vários escritórios possuem um ou mais links de conexão com clientes. Estas conexões não estão representadas no diagrama.

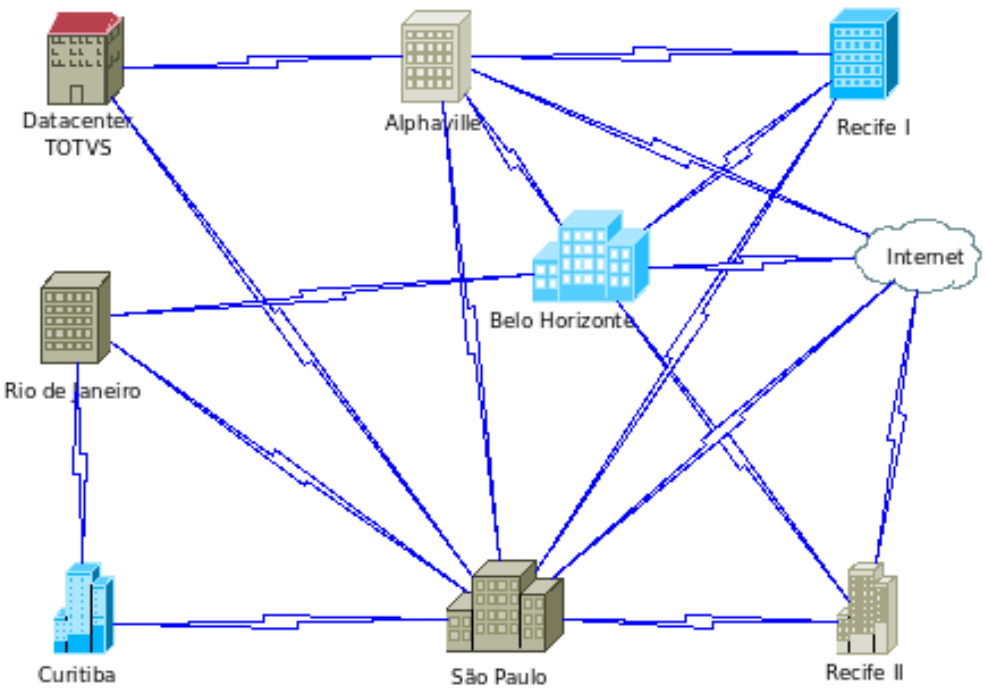

Figura 3.1: Topologia da rede WAN da empresa

Na figura 3.2 é demonstrado como o servidor de HTB foi posicionado na rede interna. Fica entre a rede interna e o firewall, conseguindo assim gerenciar

todo o tráfego originado do site de São Paulo e com destino aos sites remotos. Conforme citado anteriormente, não existe acesso à Internet por São Paulo, nem comunicações de voz com VoIP ou video-conferência que passem por estes links.

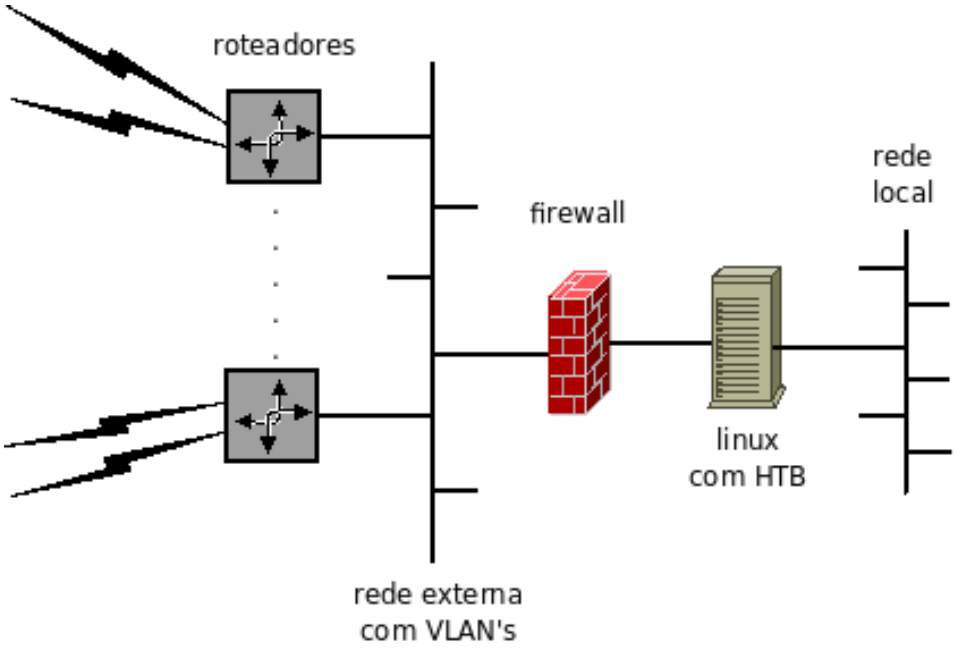

Figura 3.2: Topologia da rede LAN da empresa

3.1 Porque QoS

A ocupação da rede WAN já era uma das preocupações da equipe de tecnologia, pois já haviam reclamações esporádicas de lentidões por parte dos clientes internos e externos. A equipe de suporte da empresa, equipe que o autor faz parte, mapeou e identificou algumas características no tráfego da rede:

- Existência de aplicações antigas em duas camadas, não apropriadas para utilização em rede WAN e portanto "naturalmente lentas";
- Alguns horários apresentavam mais lentidão que outros;
- Algumas aplicações ocupavam mais banda de rede que outras;

- O maior "consumidor" de banda era o correio eletrônico.

Para complementar o estudo, várias monitorações de tráfego foram feitas utilizando as ferramentas *Open Source ntop* e *cacti*. Também foi utilizado o *Netflow Analyzer* da empresa *Manage Engine*[2].

O *Netflow Analyzer* é uma solução muito interessante para monitoração de tráfego LAN e WAN, pois recebe e processa os pacotes UDP do protocolo *netflow*, gerados por roteadores e outros componentes de rede do fabricante Cisco Systems. Cisco Netflow[3] é um protocolo desenvolvido pela Cisco com o objetivo de permitir a monitoração de banda de links ligados a seus equipamentos.

Desta monitoração foi possível identificar alguns perfis de tráfego que ocorriam sem controle e sem prioridade. A seguir são citados os mais críticos:

- Aplicações críticas, tais como "pedidos eletrônicos de autorização de compra" ou "consulta de saldo de cartão"
- Transferência de grandes arquivos para integração entre aplicativos;
- Transferência de arquivos para otimização de processos, tais como arquivos para discadores;
- Transferência de relatórios para serem impressos em outro site ou gravados em mídia ótica, para envio mais rápido a clientes;
- Transferência de arquivos de usuários, através de compartilhamento em rede *MS Windows*;
- Envio de correio eletrônico com arquivos grandes anexados;
- Acesso à Internet através dos *links* WAN privados;
- Acesso a aplicações para trabalho de produção;
- Acesso a aplicações para trabalho de retaguarda (ERP, *workflow* e similares);
- Acesso à Intranet.

[2] http://www.manageengine.com/products/netflow/

[3] Mais informações sobre Cisco Netflow podem ser obtidas no endereço eletrônico http://www.cisco.com/en/US/tech/tk812/tsd_technology_support_protocol_home.html

Somando-se ao cenário citado acima, houve o lançamento de um projeto para permitir o registro eletrônico de ponto dos funcionários, principalmente os atendentes de centrais de atendimento. Após sua implantação, os operadores de produção, atendentes e outros colaboradores, iriam registrar o ponto direto na sua estação de trabalho, e não mais nos relógios coletores de dados específicos.

Várias opções foram estudadas por uma consultoria externa, entre elas a implementação de uma rede MPLS com largura de banda maior, em substituição aos links atuais, aumento dos *links* atuais, criação de rede WAN nova apenas para este projeto e utilização de otimizadores de *WAN*.

O modelo de infra-estrutura para este projeto, apresentado à diretoria e aprovado, foi desenvolvido por uma consultoria externa e pré-aprovado pelo cliente interno (gerência de Recursos Humanos). O desenho previa a implantação de uma rede WAN separada da rede de produção, interligando todos os escritórios envolvidos. Apenas o custo de administração desta rede não foi contemplada no projeto. Como o preço da solução proposta ficou maior que o previsto, principalmente pelo custo das mensalidades dos links de dados, a equipe de suporte da empresa foi envolvida e, depois de estudos, recomendou a implementação da solução de QoS com HTB descrito abaixo.

A implementação do QoS com GNU/Linux foi escolhida inicialmente pelo baixo custo e rapidez de instalação. A contratação e instalação de novos links pelas concessionárias de telecomunicações demoraria entre 30 e 60 dias. A aquisição de soluções de QoS baseados em hardware proprietário também demoraria, pois estes equipamentos são importados e com prazo de entrega em torno de 45 dias.

Como a rede WAN entre os sites estava toda montada e funcionando, a primeira proposta técnica foi a inclusão de mais um elemento ativo entre os roteadores e o firewall, criando uma subrede DMZ ou seja, um espaço na rede protegido por dois *firewalls*. O servidor de HTB seria o segundo firewall, fazendo roteamento e *shapping* dos pacotes. Como os QoS baseados em *hardware* (*appliances*) operam em modo transparente, a equipe pesquisou em laboratório a possibilidade do Linux também conseguir fazer o HTB operar em modo *bridge*. Esta pesquisa passou inclusive pelo estudo do módulo *ebtables*[4] do Linux.

No laboratório montado, observou-se que a marcação de pacotes com o comando **tc** ou com o comando *iptables* funcionam normalmente com o Linux operando em modo *bridge*. Desta maneira ficou definido que o servidor de HTB seria instalado no site de São Paulo em modo transparente (*bridge*) antes do firewall de

[4] *ebtables* está disponível em http://ebtables.sourceforge.net/

borda ou seja, entre o *switch de core* e o *firewall*, conforme demonstrado na figura 3.2.

O site de São Paulo foi definido para receber o servidor de aplicação do serviço de Ponto Eletrônico pois é um dos sites que possuem conexão com todos os outros com redundância. Possui também vários funcionários de central de atendimento, que acessariam o novo serviço pela rede local.

A aplicação de Ponto Eletrônico foi implantada em fases, para que o projeto pudesse ser monitorado e sofresse eventuais ajustes. Até uma alteração grande de rumo poderia ser necessária, caso a solução Linux/HTB não se mostrasse competente e/ou estável o suficiente para melhorar o cenário.

A instalação do Linux com HTB foi bem documentada do ponto de vista físico, inclusive com várias fotos, de maneira que os próprios funcionários do CPD possam alterar a configuração física e desinstalar o servidor de QoS quando a monitoração interna detectar uma falha grave na solução.

A implantação do Linux com HTB foi um sucesso tanto no quesito administração do tráfego quanto em estabilidade. A aplicação de Ponto Eletrônico foi paulatinamente propagada para toda a empresa, sem a necessidade de aumento de custos com telecomunicações.

Posteriormente outro servidor com HTB foi instalado em Belo Horizonte, com o objetivo de permitir a utilização do link com a Internet para transferência de arquivos com um cliente, usando o aplicativo STCP[5].

3.2 Conhecendo o HTB

O objetivo do HTB é dividir um link de dados, seja ele de Internet ou privado, de maneira que vários tipos de tráfego de vários usuários possam utilizar o recurso, sem que um usuário consiga se sobrepor aos demais. O HTB possui várias possibilidades de configuração, incluindo priorização, empréstimo e reserva de banda. Suporta também classes filhas, virtualmente sem limite.

O HTB é um módulo do kernel do Linux (usualmente sch_htb). Para configurá-lo, criando classes e direcionando pacotes para estas classes, utiliza-se o comando "tc".

[5]STCP é desenvolvido e comercializado pela Riversoft. Riversoft pode ser encontrada em http://www.riversoft.com.br

O modulo HTB do Linux faz parte do kernel do Linux, de acordo com (DEVERA, 2003), desde a versão 2.4.20. O projeto e implementação do HTB é de 2002, de maneira que é um software bastante testado.

O módulo de kernel do HTB é carregado dinamicamente quando é feita uma referência ao mesmo, normalmente pela adição de uma fila do tipo HTB com o comando "tc".

O pacote pode ser marcado com o comando "tc" ou com o comando "iptables" usando a tabela "mangle" do mesmo. O comando "tc" está incluído no pacote "iproute", disponível para a maioria das distribuições. Em algumas distribuições o mesmo é também referenciado como "iproute2". O comando "iptables" está incluído no pacote com o mesmo nome.

3.3 Hardware e Software utilizados

O hardware utilizado em São Paulo é um servidor da marca Dell com processador Intel Xeon *dual core* de 2.0GHz de *clock*, 2Gb (*giga bytes*) de memória RAM e dois discos internos SCSI, configurados como uma unidade lógica em RAID-1 através da controladora *Dell/Perc5i*. A máquina possui também três placas de rede *ethernet* (duas ativas e uma de *backup*) e fonte de alimentação redundante. A interface *eth1* foi definida para ser ligada ao *firewall* e a *eth0* ao *switch de core*.

A distribuição utilizada para a implementação foi a *Red Hat Enterprise Linux v5.2*. Como a empresa presta serviço para muitos clientes e o nível de disponibilidade dos serviços é definida em contrato (*SLA*), o suporte da *Red Hat* é considerado essencial para uma aplicação em produção. Além da possibilidade de contratação de suporte, pesou na escolha o conhecimento interno dos profissionais da empresa com esta distribuição, pois é a distribuição usada em todos os servidores Linux com bancos de dados Oracle.

Os pacotes instalados, mandatórios para a configuração definida foram:

- *bridge-utils* - contém o utilitário *brctl*, utilizado para criar a *bridge*
- *iproute* - contém o utilitário *tc*, utilizado para criar as filas e classes
- *iptables* - contém o utilitário *iptables*, utilizado nesta implementação para marcar os pacotes, direcionando-os para as classes

Além dos pacotes citados acima, foram instalados também os utilitários *iftop*[6], o *iptraf*[7] e o *tc-viewer*[8]. Os dois primeiros exibem os maiores fluxos de tráfego *ethernet*, permitindo ordenação e filtros. Possuem uma abordagem muito similar. O *tc-viewer* é especifico para monitoração do HTB e exibe o tráfego médio em cada fila. Estes programas são de visão instantânea ou seja, não guardam informações para análise posterior. Outro utilitário instalado foi o *nload*[9]. O *nload* mostra graficamente o tráfego por dispositivo de rede, mas em uma janela de terminal.

No servidor foi também instalado o *ntop* com várias instâncias, cada uma monitorando um fluxo de dados diferente, permitindo um acompanhamento "a posteriori" do fluxo das filas mais importantes.

A instalação dos pacotes foi feita com o comando *rpm -iv <nome-do-pacote>*. Dos utilitários citados, o *iptraf*, *nload* e o *iftop* são disponibilizados através dos fontes. A seguinte forma de compilação, tradicional do Linux, deve ser utilizada para gerar o binário dos mesmos:

- download do pacote
- descompactação em um diretório de trabalho
- posicionar no diretório do utilitário
- ./configure
- make
- make install

O *tc-viewer* é um *script* em *perl* e não necessita de instalação. Basta copiá-lo para um diretório de binários que esteja no caminho de execução (variável de ambiente *PATH*). Normalmente é copiado para */usr/local/bin*.

Os utilitários respondem ao parâmetro "-h" ou ao parâmetro "–help" e fornecem informações detalhadas para seu uso. Somente o *tc-viewer* obriga a utilização de, no mínimo, o parâmetro interface. Os demais executam sem parâmetros e assumem valores *default*.

[6] *iftop* está disponível para download em http://www.ex-parrot.com/ pdw/iftop
[7] *iptraf* está disponível para download em http://iptraf.seul.org
[8] *tc-viewer* está disponível para download em http://pawilcz.eu/tc-viewer/tc-viewer.html
[9] *nload* está disponível para download em http://www.roland-riegel.de/nload/

3.4 Configuração do servidor de HTB

A configuração do servidor de *HTB* foi separada em três *scripts*. Os *scripts* são executados na inicialização do servidor através de sua inclusão no arquivo */etc/rc.local*, disponível na distribuição Red Hat e derivadas. Este arquivo é chamado automaticamente na inicialização do Linux, após a execução dos scripts do diretório /etc/init.d.

Nas listagens de *scripts* e saídas de comandos deste capítulo, quando não houver prejuízo da informação ou do entendimento, haverá uma redução das linhas exibidas, com o objetivo de melhorar a apresentação do documento. Esta supressão de linhas é registrada pela inclusão de reticências (...) nas listagens. As listagens completas estão apresentadas no apêndice.

3.4.1 Criação da bridge

Para a criação da *bridge* foi definido o *script criabr.sh*. O *script* é muito simples e seu código é apresentado na figura 3.3. O endereço *IP* atribuído à bridge é utilizado para fins de monitoração e acesso remoto. Ele não interfere nas regras nem nas classes do *HTB*.

```
1  #!/bin/sh
2  #Script criabr.sh
3  #
4  #Script para criacao de bridge para HTB
5  #Este script nao recebe parametros
6
7  brctl delbr br0
8
9  brctl addbr br0
10 brctl addif br0 eth0
11 brctl addif br0 eth1
12
13 ifconfig br0 192.168.200.201/24 up
```

Figura 3.3: Script **criabr.sh**

Na linha 5 da figura 3.3 é criada a *bridge* de nome *br0*. Nas linhas 6 e 7 são adicionadas as interfaces *eth0* e *eth1* à *bridge*. Na linha 8 a bridge é ativada e recebe o endereço IP 191.168.200.201. A interface *eth1* é ligada ao *firewall* e a *eth0* é ligada ao *switch* interno.

3.4.2 Criação das filas e classes

Para a definição das filas e classes foi criado o *script* **criafilas.sh**. O *script* também é muito simples. Fragmentos do seu código são apresentados nas figuras 3.4 e 3.5. Foi definido que as regras da interface *eth1* são consideradas de saída e as da *eth0* como de entrada.

```
#!/bin/sh
#Script para criacao de filas e classes HTB
#Este script nao recebe parametros

#Limpa as filas previamente criadas
tc qdisc del dev eth0 root
tc qdisc del dev eth1 root

#######Cria classe raiz do controle sentido eth0 => eth1 ↘
→######## 10=out
tc qdisc add dev eth1 root handle 10: htb default 99

## Recife II #####
tc class add dev eth1 parent 10:  classid 10:1  htb rate 2048↘
→kbit ceil 2048kbit
tc class add dev eth1 parent 10:1 classid 10:11 htb rate  700↘
→kbit ceil 2048kbit prio 1
tc class add dev eth1 parent 10:1 classid 10:12 htb rate  400↘
→kbit ceil 2048kbit prio 2
tc class add dev eth1 parent 10:1 classid 10:13 htb rate  256↘
→kbit ceil 1536kbit prio 4
tc class add dev eth1 parent 10:1 classid 10:14 htb rate  200↘
→kbit ceil 1536kbit prio 3
...
...
## Default de saida #####
tc class add dev eth1 parent 10: classid 10:99 htb  rate   64↘
→kbit ceil 1024kbit prio 5
```

Figura 3.4: Script *criafilas.sh* parte 1

A primeira classe que deve ser configurada para uma determinada interface, a "root". Esta classe determina que o tratamento será com HTB, uma classe que suporta classes inferiores ou sub-classes. É possível que as classes internas sejam criadas com outros métodos de QoS, podendo o administrador aproveitar melhores características de cada solução. Na criação da classe "root" é especificado o "id" e qual será a classe *default*, para a qual serão enviados os pacotes não marcados. No exemplo, foi utilizado a classe *10* para saída e a classe **11** para entrada.

```
#######Cria classe raiz do controle sentido eth1 => eth0 ↘
→######## 11=in
tc qdisc add dev eth0 root handle 11: htb default 99

#Internet
tc class add dev eth0 parent 11:  classid 11:8  htb rate 3072↘
→kbit ceil 3072kbit
tc class add dev eth0 parent 11:8 classid 11:81 htb rate  512↘
→kbit ceil 1536kbit
tc class add dev eth0 parent 11:8 classid 11:82 htb rate 1024↘
→kbit ceil 3072kbit

## Recife II #####
tc class add dev eth0 parent 11:  classid 11:1  htb rate 2048↘
→kbit ceil 2048kbit
tc class add dev eth0 parent 11:1 classid 11:11 htb rate  700↘
→kbit ceil 2048kbit prio 1
tc class add dev eth0 parent 11:1 classid 11:12 htb rate  400↘
→kbit ceil 2048kbit prio 2
tc class add dev eth0 parent 11:1 classid 11:13 htb rate  256↘
→kbit ceil 1536kbit prio 3
tc class add dev eth0 parent 11:1 classid 11:14 htb rate  200↘
→kbit ceil 1536kbit prio 4
...
...
## Default de entrada #####
tc class add dev eth0 parent 11: classid 11:99 htb  rate   64↘
→kbit ceil 1024kbit prio 5
```

Figura 3.5: Script *criafilas.sh* parte 2

As entradas de classes são no formato *x:y* (para HTB e para outras *qdisc*) onde **x** é um inteiro identificador da *qdisc* e **y** é um inteiro que identifica a classe desta *qdisc*. Quando o comando *tc* especifica uma *qdisc*, o valor **y** deve ser zero. Quando especifica uma classe, o valor de *y* deve ser maior que zero.

Na implementação atual, logo após a criação da classe *root* são criadas subclasses para cada link que deseja fazer o QoS do mesmo. Não foram criadas classes para todos os links. Uma classe se liga à classe *root* ou a uma subclasse, através do parâmetro *parent* (pai). O identificador da classe é especificado no parâmetro *classid* e deve ser um inteiro, separado da *qdisc* por dois pontos. A definição de uma seqüência lógica como a utilizada aqui serve para entendimento das regras, mas não é exigida pelo HTB. Qualquer classe pode ser filha de qualquer outra.

Cada uma das classes pai (ligada diretamente à *root*), deve ter o valor do parâmetro de reserva de banda idêntico ao do limite de utilização. O valor para reserva de banda é especificado pelo parâmetro *rate* e o valor para limite de utilização pelo parâmetro *ceil*.

O comando **tc** aceita diversas unidades para a taxa. Na figura 3.6 é mostrada as possibilidades para especificação das unidades para a taxa, tanto para *rate* quanto para *ceil*.

Figura 3.6: Unidades para especificação de taxas no comando **tc**

mnemônico	**especificação**
`kbps`	quilo bytes por segundo
`mbps`	mega bytes por segundo
`kbit`	quilo bits por segundo
`mbit`	mega bits por segundo
`bps`	bits por segundo

A literatura menciona a possibilidade de utilização de banda acima da garantida como "empréstimo". Em HTB, esta operação pode ser considerada como "doação" porque a classe não devolve a banda recebida para a sua classe pai. Este empréstimo somente acontece se as classes de prioridade maior (numeração mais baixa) não necessitar de banda no momento.

A priorização é configurada através da utilização do parâmetro *prio* na linha de comando. A priorização só demonstra sua eficácia quando duas ou mais classes solicitarem "empréstimo". A classe com maior prioridade (número menor) receberá a sobra primeiro. Somente quando esta não tiver pacotes para enviar é que a sobra será ofertada para a classe de menor prioridade.

Os últimos dois parâmetros que podem ser utilizados no comando **tc** quando a *qdisc* for *HTB* são *burst* e *cburst*. Estes parâmetros estão interrelacionados e não é comum a alteração dos mesmos. Para explicá-los, é necessário explicar um pouco do funcionamento do hardware.

O envio do pacote, seja em *ethernet* ou outro meio, é feito um a um e, com certeza, na velocidade de envio do hardware. Se o hardware é uma placa *ethernet* de 100Mbps, cada bit de dados sairá somente nesta velocidade, mesmo que o destino final seja um link WAN de 128kbps através de um roteador. Mesmo que hajam vários fluxos TCP/IP trafegando dados "simultaneamente", a simultaneidade é aparente pois a transmissão física final será com pulsos elétricos (ou

luminosos) e o sinal é enviado sempre de forma serial. O que o software faz é enviar uma pequena rajada de pacotes da classe A, depois outra rajada da classe B, assim sucessivamente até terminarem as classes. Depois volta novamente a transmitir outra rajada de pacotes da classe A.

Os parâmetros *burst* e *cburst* controlam a quantidade de bytes que serão enviados para a rede, enquanto o *HTB* estiver transmitindo para uma classe específica. O *burst* especifica a quantidade de *bytes* que será enviada na velocidade "*ceil*" enquanto *cburst* especifica a quantidade de bytes que será enviada na velocidade do hardware. *HTB* tenta se lembrar do *burst* (rajada) de uma classe que foi utilizado por seus filhos por até um minuto.

3.4.3 Consultas às classes e *qdiscs*

A configuração corrente do HTB pode ser consultada com o próprio comando **tc**. A figura 3.7 mostra a saída do comando, no servidor de *HTB* da empresa. Caso seja desejável, pode incluir o *device*, conforme figura 3.8.

```
# tc qdisc show
qdisc htb 11: dev eth0 r2q 10 default 0 direct_packets_stat ↘
→88627791
qdisc htb 10: dev eth1 r2q 10 default 0 direct_packets_stat ↘
→2021674671
```

Figura 3.7: Consulta às *qdiscs*

```
# tc qdisc show dev eth1
qdisc htb 10: dev eth1 r2q 10 default 0 direct_packets_stat ↘
→2021674671
```

Figura 3.8: Consulta à *qdisc* especificando dispositivo

O usuário pode ainda solicitar informações sobre as estatísticas, usando o parâmetro "-s". Exemplo desta opção está na figura 3.9. As estatísticas são referentes à *qdisc*. O número de "direct_packet_stat", conforme (DEVERA, 2003), informa a quantidade de pacotes enviados através da fila. Os demais valores são auto-explicativos.

O comando **tc** permite também a consulta das classes e suas estatísticas. As opções são muito similares às consultas para as *qdiscs*. Alguns exemplos estão incluídos nas figuras 3.10, 3.11 e 3.12.

```
# tc -s qdisc show
qdisc htb 11: dev eth0 r2q 10 default 0 direct_packets_stat ↘
→90718338
 Sent 5524354409578 bytes 2507881780 pkt (dropped 0, ↘
 →overlimits 466788941 requeues 23)
 rate 0bit 0pps backlog 0b 0p requeues 23
qdisc htb 10: dev eth1 r2q 10 default 0 direct_packets_stat ↘
→2023798882
 Sent 6198568475041 bytes 2411290971 pkt (dropped 0, ↘
 →overlimits 840149492 requeues 27)
 rate 0bit 0pps backlog 0b 0p requeues 27
```

Figura 3.9: Consulta a estatísticas de *qdiscs*

```
# tc  class show dev eth1
class htb 10:11 parent 10:1 prio 1 rate 600000bit ceil 1536↘
→Kbit burst 1674b cburst 1791b
class htb 10:22 parent 10:2 prio 2 rate 424000bit ceil 1024↘
→Kbit burst 1652b cburst 1728b
class htb 10:1 root rate 1536Kbit ceil 1536Kbit burst 1791b ↘
→cburst 1791b
class htb 10:32 parent 10:3 prio 2 rate 768000bit ceil 1024↘
→Kbit burst 1695b cburst 1728b
class htb 10:2 root rate 1024Kbit ceil 1024Kbit burst 1728b ↘
→cburst 1728b
class htb 10:31 parent 10:3 prio 1 rate 256000bit ceil 1024↘
→Kbit burst 1632b cburst 1728b
class htb 10:12 parent 10:1 prio 2 rate 936000bit ceil 1536↘
→Kbit burst 1716b cburst 1791b
...
...
```

Figura 3.10: Fragmento de consulta a classes

3.5 Marcação dos pacotes

Para que o HTB desempenhe sua função corretamente, os pacotes IP precisam ser marcados e direcionados para uma classe específica. Um pacote não marcado será encaminhado para a fila *default* da interface pela qual o mesmo sairá.

Os pacotes podem ser marcados com o comando **tc** ou com o comando *iptables*, utilizando a tabela *mangle* do mesmo. Na empresa, a equipe de suporte optou por utilizar o *iptables*. Um exemplo para marcação de pacotes com o comando *tc* é

```
# tc -s  class show dev eth1
class htb 10:11 parent 10:1 prio 1 rate 600000bit ceil 1536↘
→Kbit burst 1674b cburst 1791b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 22333 ctokens: 9333

class htb 10:1 root rate 1536Kbit ceil 1536Kbit burst 1791b ↘
→cburst 1791b
 Sent 12813716336 bytes 136118726 pkt (dropped 0, overlimits 0↘
 → requeues 0)
 rate 4720bit 4pps backlog 0b 0p requeues 0
 lended: 19063671 borrowed: 0 giants: 0
 tokens: -5132 ctokens: -5132
...
...
```

Figura 3.11: Fragmento de consulta a classes com estatísticas

```
# tc -s -d class show dev eth1
class htb 10:11 parent 10:1 prio 1 quantum 7500 rate 600000bit↘
→ ceil 1536Kbit burst 1674b/8 mpu 0b overhead 0b cburst 1791b↘
→/8 mpu 0b overhead 0b level 0
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 22333 ctokens: 9333

class htb 10:1 root rate 1536Kbit ceil 1536Kbit burst 1791b/8 ↘
→mpu 0b overhead 0b cburst 1791b/8 mpu 0b overhead 0b level 7
 Sent 12813716336 bytes 136118726 pkt (dropped 0, overlimits 0↘
 → requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 19063671 borrowed: 0 giants: 0
 tokens: -5132 ctokens: -5132
...
...
```

Figura 3.12: Fragmento de consulta a classes com estatísticas detalhadas

demonstrada na figura 3.13. As linhas de da figura 3.14 atingem o mesmo objetivo, porém usando o comando *iptables*.

```
tc filter add dev eth1 protocol ip parent 10:0 prio 1 u32 ↘
→match ip dst 192.168.20.0/24 match ip sport 80 0xffff flowid ↘
→10:21
tc filter add dev eth1 protocol ip parent 10:0 prio 1 u32 ↘
→match ip dst 192.168.30.0/24 match ip sport 80 0xffff flowid ↘
→10:31
```

Figura 3.13: Filtros que direcionam pacotes para classes HTB com **tc**

```
iptables -t mangle -A POSTROUTING -p tcp -d 192.168.20.0/24 --↘
→sport 80 -j CLASSIFY --set-class 10:21
iptables -t mangle -A POSTROUTING -p tcp -d 192.168.30.0/24 --↘
→sport 80 -j CLASSIFY --set-class 10:31
```

Figura 3.14: Direcionando pacotes para classes com **iptables**

Na figura 3.15 está incluído um fragmento do *script* **marcacao.sh**. O primeiro comando válido do **script** limpa a tabela do *iptables*. A seguir, são incluídas várias regras para marcação dos pacotes, conforme definição do administrador. Todas as opções de seleção de pacotes disponíveis no *iptables*, tais como *multiport*, *protocolo*, *origem*, *destino* e etc. O parâmetro *-j* do *iptables* recebe a constante *CLASSIFY* e a seguir, na mesma linha, deve ser informada para qual classe o pacote deve ser encaminhado. É possível observar na figura 3.15 que o autor do *script* incluiu todas linhas de marcação de um determinado site agrupadas, para facilitar a manutenção do *script*.

Uma observação importante sobre o comportamento do *iptables* quando utilizado o destino *CLASSIFY* em uma regra. De maneira análoga ao destino *LOG*, o *CLASSIFY* não interrompe o processamento das demais regras. É necessário então cautela na definição da sequência das regras. Se um pacote for marcado em uma regra, e posteriormente for marcado por outra, o HTB o tratará na classe para a qual ele foi marcado por último. Esta análise deve ser feita considerando a numeração das regras, e não a seqüência que as mesmas aparecem no *script*.

3.6 Monitoração do *QoS*

Os comandos citados anteriormente (*ntop, iftop, iptraf, tc-viewer* e *nload*) são usados para monitoração do funcionamento do HTB e da *bridge*. O mais utilizado é o *tc-viewer.pl* pois ele realmente é dedicado e monitorar as classes.

```
#!/bin/sh
#Script marcacao.sh

#Limpa tabels mangle do iptables
iptables -t mangle -F

##### MARCA PACOTES #####

##### Recife ############

iptables -t mangle -A POSTROUTING -p tcp -s 10.181.0.0/21 -d ↘
→191.167.4.25 -j CLASSIFY --set-class 11:21
iptables -t mangle -A POSTROUTING -p tcp -s 191.167.4.25 -d ↘
→10.181.0.0/21 -j CLASSIFY --set-class 10:21

iptables -t mangle -A POSTROUTING -p all -s 10.181.0.0/21 -d !↘
→ 191.167.4.25 -j CLASSIFY --set-class 11:22
iptables -t mangle -A POSTROUTING -p all -s ! 191.167.4.25 -d ↘
→10.181.0.0/21 -j CLASSIFY --set-class 10:22
...
...
```

Figura 3.15: Fragmento do *script* **marcacao.sh**

É comum no ambiente da empresa, a utilização do **tc-viewer** com a seguinte linha de comando:

- tc-viewer.pl -iface eth1 –zero –colors –unit=kbit –timer=5

Depois de cinco segundos, a tela demonstrada na figura 3.16 é exibida, sendo atualizada a cada cinco segundos.

As opções definem que o **tc-viewer** monitore a interface de rede eth1, mostre as classes com zero de tráfego, exiba a informação em cores, use *quilo bit* como unidade de exibição e atualize a tela a cada 5 segundos. O **tc-viewer** pode usar um arquivo de configuração para as opções. Este arquivo pode ser passado usando o parâmetro "–conf". Através do arquivo de parâmetro, várias opções podem ser definidas, entre elas um apelido para as classes numéricas.

Em alguns momentos não é possível ver o tráfego pelo **tc-viewer**, pois ele mostra a estatística atual. Se o tráfego for muito pequeno, é possível que o mesmo seja arredondado para zero. Nestas situações podemos usar o comando **iptables -L -t mangle -v** para ver a contagem de pacotes que o *iptables* marcou por regra,

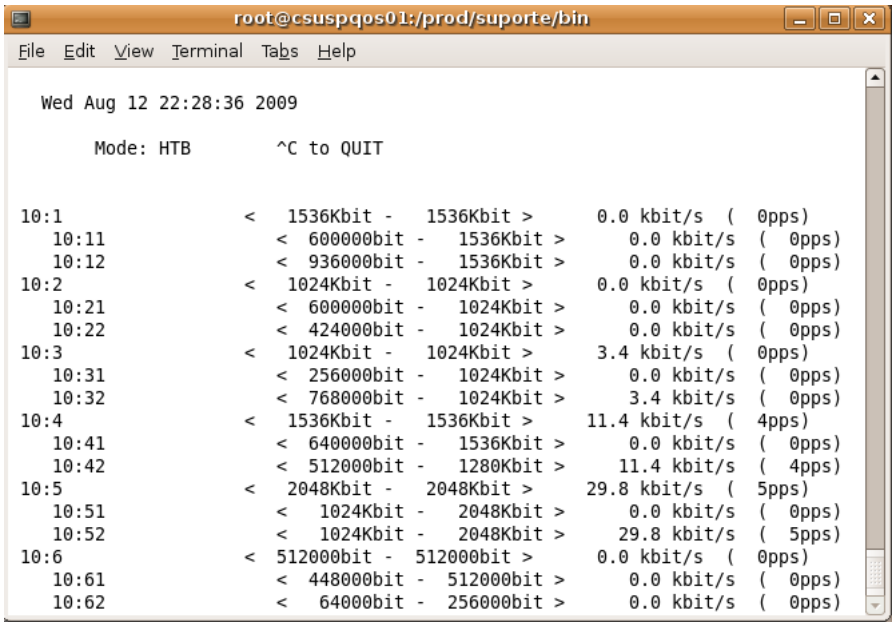

Figura 3.16: Tela do comando **tc-viewer**

sabendo assim, indiretamente, o funcionamento do HTB. Um exemplo da saída deste comando é incluído na figura 3.17.

Uma tela do comando **iptraf** é demonstrada na figura 3.18. Na figura 3.19 pode ser vista a tela do comando **nload**. Na figura 3.20 é exibida uma tela do comando **iftop**.

```
...
...
Chain POSTROUTING (policy ACCEPT 74G packets, 11T bytes)
 pkts bytes target     prot opt in     out     source ↘
→               destination
    0     0 CLASSIFY   tcp  --  any    any     191.165.0.0/16 ↘
    →       191.167.4.25         CLASSIFY set 11:11
    0     0 CLASSIFY   tcp  --  any    any     191.167.4.25 ↘
    →         191.165.0.0/16       CLASSIFY set 10:11
    0     0 CLASSIFY   all  --  any    any     191.165.0.0/16 ↘
    →       !191.167.4.25          CLASSIFY set 11:12
 317M   29G CLASSIFY   all  --  any    any    !191.167.4.25 ↘
 →         191.165.0.0/16       CLASSIFY set 10:12
 143M   22G CLASSIFY   tcp  --  any    any     10.181.0.0/21 ↘
 →        191.167.4.25          CLASSIFY set 11:21
 116M   43G CLASSIFY   tcp  --  any    any     191.167.4.25 ↘
 →         10.181.0.0/21        CLASSIFY set 10:21
 294M   72G CLASSIFY   all  --  any    any     10.181.0.0/21 ↘
 →        !191.167.4.25         CLASSIFY set 11:22
 255M  248G CLASSIFY   all  --  any    any    !191.167.4.25 ↘
 →         10.181.0.0/21        CLASSIFY set 10:22
...
...
```

Figura 3.17: Fragmento do comando **iptables -L -t mangle -v**

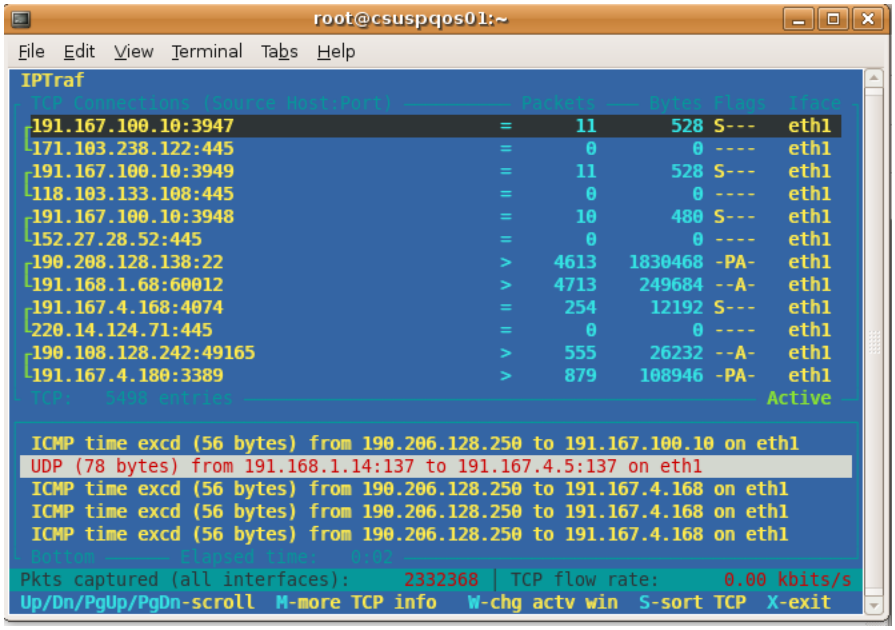

Figura 3.18: Tela do comando **iptraf**

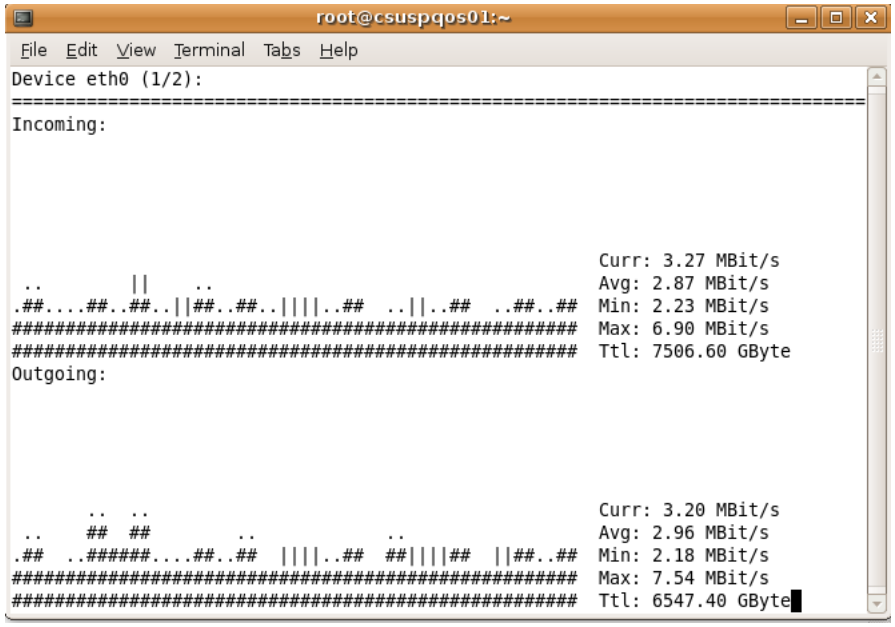

Figura 3.19: Tela do comando **nload**

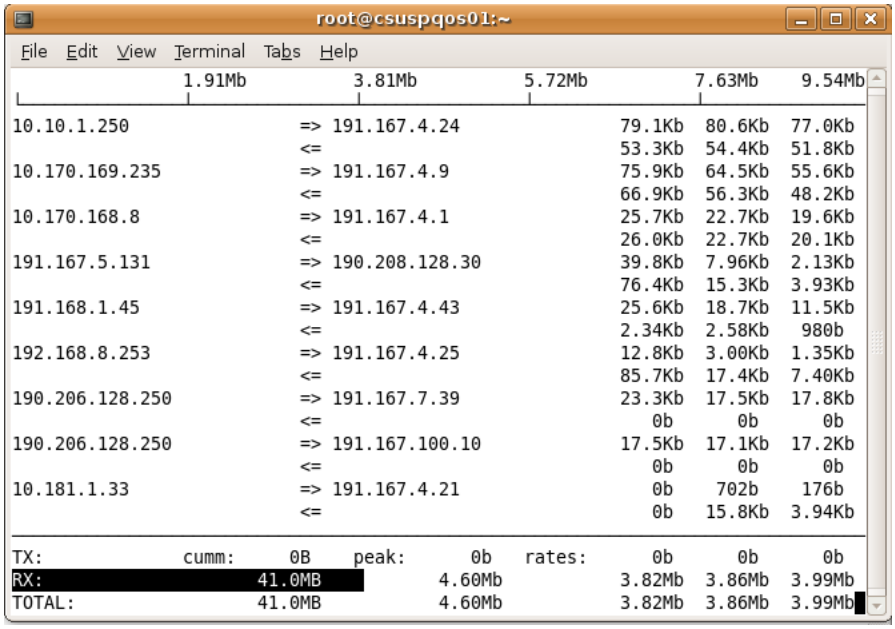

Figura 3.20: Tela do comando **iftop**

Capítulo 4

Testes realizados

O objetivo da realização destes testes é demonstrar o funcionamento correto do HTB, utilizando um ambiente controlado. Para atender a este objetivo, foi criado um laboratório com hardware, software e metodologia para os testes.

4.1 Ambiente de testes

O laboratório utilizado para os testes consiste em um equipamento de marca HP com processador da fabricante AMD e modelo "Turion 64 X2" de núcleo duplo, 1.9GHz e 4Gb de memória real, rodando Linux Ubuntu 8.04 de 32 bits. Este hardware foi utilizado como *host* de virtualização. A solução de virtualização utilizada foi VirtualBox[1], da Sun Microsystems.

Sobre o VirtualBox foram instalados duas outras máquinas com o mesma versão de Linux que a do *host*, uma agindo como *"bridge"* e o outra como *"server"*. Na máquina *"bridge"* está configurada a solução de HTB em modo *bridge*. Ver figura 4.1. A versão do VirtualBox utilizada não permite que cada máquina virtual utilize mais que um processador. Desta forma, existe um balanceamento de performance entre as três máquinas envolvidas, baseado no *task scheduler* do sistema operacional Linux do computador (*host*).

Gerar tráfego controlado com aplicações convencionais não é uma tarefa simples. Para a geração de tráfego controlado para este trabalho, foram escritos dois programas em linguagem C, chamados **socketclient** e **socketserver**. Estes progra-

[1] Disponível para download em www.sun.com/software/products/virtualbox/get.jsp

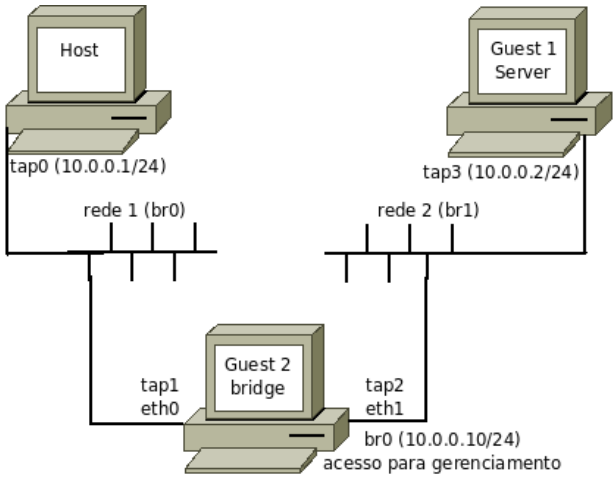

Figura 4.1: Topologia do laboratório de testes

mas têm a capacidade de efetuar a comunicação em até oito portas TCP ao mesmo tempo, podendo qualquer uma destas portas serem utilizadas para transmissão ou para recepção. A comunicação é feita em blocos, cujo tamanho padrão é de 1024 bytes.

O grande diferencial deste conjunto de programas em relação a alguns outros que se propõem a geração de tráfego, como por exemplo o *iperf*[2], é que ester permitem a utilização de portas TCP/IP especificadas individualmente para cada conexão e tambem permite configuração de retardo no inicio da comunicação.

O programa **socketserver** deve ser chamado primeiro, com até oito portas TCP como parâmetros. Ficará escutando indefinidamente estas portas. A comunicação de cada porta roda em uma *thread* independente. Depois que uma comunicação termina, a *thread* volta a escutar a mesma porta, aguardando o inicio de uma nova comunicação.

Enquanto o programa **socketserver** está em execução, o **socketclient** pode então ser executado. Este programa deve receber o endereço IP onde o **socketserver** está em execução (parâmetro -s) e, no mínimo, uma opção de comunicação, que pode ser "t" (para transmissão) ou "r" (para recepção) seguido da porta, quantidade de blocos e tempo de espera. A figura 4.2 mostra uma chamada típica dos programas. Mais detalhes sobre outras opções dos programas estão disponíveis na seção A.3 do apêndice deste documento. Na figura 4.2, a primeira linha mostra a

[2]disponível na Internet, no endereço http://sourceforge.net/projects/iperf/

chamada do **socketserver** para escutar as portas 20, 21, 22, 80 e 443 do equipamento, em todos os IPs configurados no mesmo. Na linha seguinte é apresentada a chamada do **socketclient** para se conectar no IP 10.0.0.1 e abrir três *threads*, a primeira recebendo 3.150 blocos na porta 20, a segunda recebendo 1.350 blocos na porta 80 após esperar 120 segundos e a terceira transmitindo 1.000 blocos pela porta 22.

```
./socketserver 20 21 22 80 443 -w saida.log
./socketclient -s 10.0.0.1 -r 20 3150 0 -r 80 1350 120 -t 22 ↘
→1000 0
```

Figura 4.2: Exemplo de chamada dos programas de teste

A criação da qdisc e das classes do HTB para os testes pode ser consultada na figura 4.3. As linhas 1 e 2 limpam as configurações anteriores. Na linha 4 é criada a *qdisc* do tipo HTB, com identificador 11, definindo que o tráfego não classificado deve ser enviado para a classe 11:299. Na linha 6 é criada uma classe com identificador 11:1 com limite de 100kbps (100 kilobits por segundo), para simular um link lento entre as máquinas. Uma configuração similar é sempre necessária para informar ao HTB o tamanho do link disponível, para que ele possa gerenciar o tráfego corretamente.

```
1  iptables -t mangle -F
2  tc qdisc del dev eth1 root
3
4  tc qdisc add dev eth1 root handle 11: htb default 299
5
6  tc class add dev eth1 parent 11:   classid 11:1   htb rate 100↘
   →kbit ceil 100kbit burst 1 cburst 1
7  tc class add dev eth1 parent 11:1  classid 11:11  htb rate  10↘
   →kbit ceil 100kbit prio 3 burst 1 cburst 1
8  tc class add dev eth1 parent 11:1  classid 11:12  htb rate  30↘
   →kbit ceil 100kbit prio 1 burst 1 cburst 1
9  tc class add dev eth1 parent 11:1  classid 11:20  htb rate  55↘
   →kbit ceil 100kbit prio 2 burst 1 cburst 1
10 tc class add dev eth1 parent 11:20 classid 11:201 htb rate  35↘
   →kbit ceil 100kbit prio 2 burst 1 cburst 1
11 tc class add dev eth1 parent 11:20 classid 11:202 htb rate  20↘
   →kbit ceil 100kbit prio 2 burst 1 cburst 1
12 tc class add dev eth1 parent 11:1  classid 11:299 htb rate   5↘
   →kbit ceil  25kbit prio 4 burst 1 cburst 1
```

Figura 4.3: Configuração do HTB de testes

Nas linhas 7, 8 e 9 são criadas as classes 11:11, 11:12 e 11:20, respectivamente com 10kbps, 30kbps e 55kbps de banda reservada. Nas linhas 10 e 11 são criadas as classes 11:201 e 11:202 que são classes filhas da 11:20. Como a classe 11:20 possui uma banda reservada de 55kbps, a soma das bandas de suas classes filhas não deve ultrapassar este valor. No exemplo, as classes filhas possuem, respectivamente, 35kbps e 20kbps. Na linha 12 é criada a última classe do laboratório, a classe 11:299 que receberá o tráfego não classificado. Esta classe possui uma banda garantida de apenas 5kbps.

Nas figura 4.4 estão as linhas que classificam o tráfego para cada classe, utilizando a tabela *mangle* do *iptables*.

Todas as classes podem utilizar toda a capacidade do tráfego através de empréstimo, exceto a classe "default", que está limitada a um tráfego máximo de 25kbps. O empréstimo é controlado pela disponibilidade e o acesso é concedido para a classe inferior baseado na prioridade. Neste exemplo, a classe 11:11 possui prioridade 3, a 11:12 prioridade 1, as 11:201 e 11:202 possuem prioridade 2 e a 11:299 possui prioridade 4.

As prioridades acima citadas foram definidas para quatro níveis: alta, normal, media e baixa. O tráfego SSH (porta 22) foi definido como prioridade alta e será direcionado para a classe 11:12. O trafego de HTTP e HTTPS (respectivamente portas 80 e 443) serão direcionados paras a classes 11:201 e 11:202 com prioridades normal e média respectivamente. O tráfego de FTP-DATA e FTP (portas 20 e 21 respectivamente) terá prioridade média e será encaminhado para a classe 11:11. Finamente, o tráfego não classificado fluirá automaticamente pela classe 11:299 com prioridade baixa.

```
1  iptables -t mangle -A POSTROUTING -p tcp -s 10.0.0.1/32 --↘
   →sport 20 -j CLASSIFY --set-class 11:11
2  iptables -t mangle -A POSTROUTING -p tcp -s 10.0.0.1/32 --↘
   →sport 21 -j CLASSIFY --set-class 11:11
3  iptables -t mangle -A POSTROUTING -p tcp -s 10.0.0.1/32 --↘
   →sport 22 -j CLASSIFY --set-class 11:12
4  iptables -t mangle -A POSTROUTING -p tcp -s 10.0.0.1/32 --↘
   →sport 80 -j CLASSIFY --set-class 11:201
5  iptables -t mangle -A POSTROUTING -p tcp -s 10.0.0.1/32 --↘
   →sport 443 -j CLASSIFY --set-class 11:202
```

Figura 4.4: Configuração do HTB de testes - continuação

4.2 Teste 1 - QoS de FTP e HTTP

Esta teste utiliza dois fluxos TCP, um na porta 20 (FTP-DATA) e outro na porta 80 (HTTP). A quantidade de blocos de dados de cada fluxo foi calculada para que o primeiro tráfego demore aproximadamente seis minutos e que o segundo demore aproximadamente dois minutos, iniciando dois minutos após o inicio do primeiro.

O objetivo deste teste é demonstrar que o HTB permitirá o FTP-DATA utilizar toda a banda disponível, enquanto não existir outro tráfego com maior prioridade. O programa *socketclient*, ao ser iniciado, começa o tráfego na porta 20 (FTP-DATA). Como existirá somente este tráfego, o mesmo ocupará toda a banda disponível. Após dois minutos, a *thread* que trafega na porta 80 (HTTP) inicia o seu tráfego e neste momento o HTB passará toda a banda disponível para o mesmo pois o tráfego HTTP (neste exemplo) tem mais prioridade do que o tráfego de FTP-DATA. o tráfego de FTP-DATA continua, durante o período de tráfego do HTTP, somente com sua banda reservada (10kbps). Após o término do tráfego na porta de HTTP, o HTB passa novamente para a porta de FTP toda a banda da classe 10:1. Na figura 4.5, os comandos utilizados para este teste podem ser vistos. O resultado do tráfego é demonstrado graficamente na figura 4.6. Na figura 4.7 é demonstrado uma tela do comando **tc-viewer.pl** capturada quando o fluxo de tráfego na porta HTTP estava ocorrendo em paralelo com o tráfego na porta FTP-DATA.

```
1  ./socketserver 20 21 22 80 443 –w saida.log
2  ./socketclient –s 10.0.0.1 –r 20 3150 0 –r 80 1350 120
```

Figura 4.5: Linhas de comandos utilizadas no teste 1

A quantidade de blocos de tráfego submetido em cada um das portas foi calculado utilizando a duração esperada do fluxo e a taxa que o HTB permite passar para a porta especifica. Neste exemplo, nos primeiros 120 segundos o FTP trafega a 100kbps, dando um total de 1500 blocos de 1024 bytes (fórmula 4.1).

$$blocos = \frac{100kbps * 120segundos}{1024bytes * 8bits} \tag{4.1}$$

Na segunda parte do teste (mais 120 segundos), teremos o HTTP trafegando a 90kbps e o FTP a 10kbps. Para o primeiro fluxo serão necessários 1350 blocos (fórmula 4.2) e para o segundo serão necessários 150 blocos (fórmula 4.3)

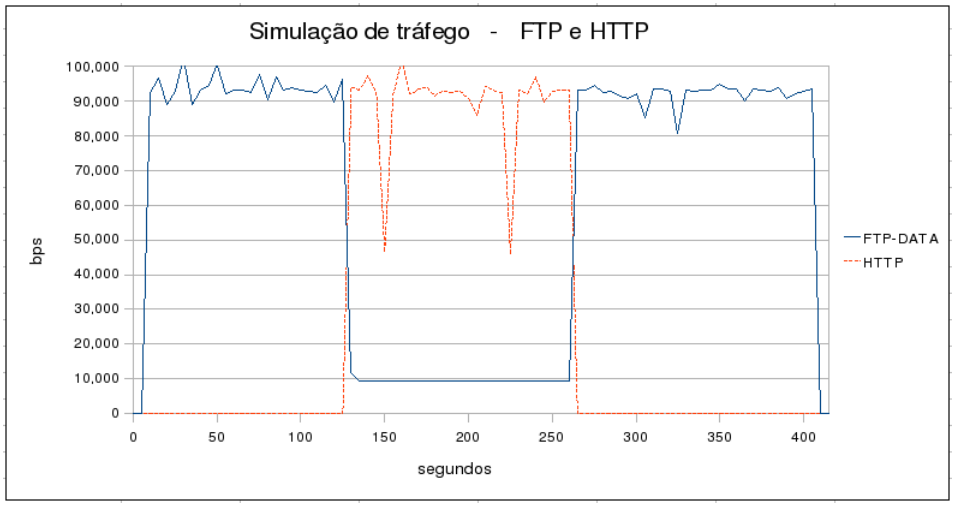

Figura 4.6: Teste 1 - Tráfego com QoS

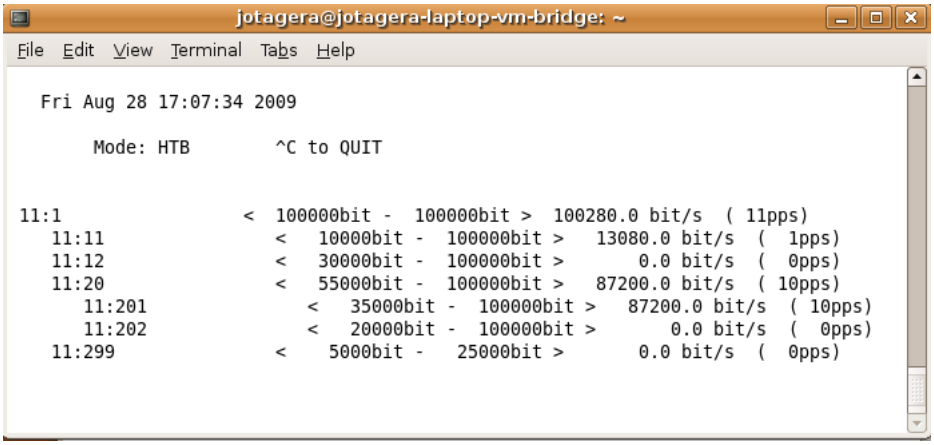

Figura 4.7: Teste 1 - Tráfego com QoS, visão do **tc-viewer**

$$blocos = \frac{90kbps * 120segundos}{1024bytes * 8bits} \tag{4.2}$$

$$blocos = \frac{10kbps * 120segundos}{1024bytes * 8bits} \tag{4.3}$$

Na terceira parte do teste, a quantidade de blocos de FTP é a mesma da primeira parte. No total, são necessários 3.150[3] blocos de FTP e 1.350 blocos de HTTP.

4.3 Teste 2 - FTP e HTTP sem QoS

Este teste tem o objetivo de demonstrar o comportamento da rede quando vários fluxos de dados ocorrem no mesmo intervalo de tempo, sem gerência de banda.

Sem controle nem priorização, os fluxos tendem a automaticamente dividir o tráfego entre eles, de maneira que nenhum ganhará mais ou menos banda. A figura 4.8 demonstra o tráfego de dois fluxos sem controle. Inicialmente o fluxo de FTP-DATA ocupa toda a banda de 100kbps. Após aproximadamente dois minutos tem inicio o tráfego de HTTP. Como o programa simulador usa os mesmos tamanhos de mensagem e o mesmo protocolo, a ocupação da banda será dividida entre os fluxos.

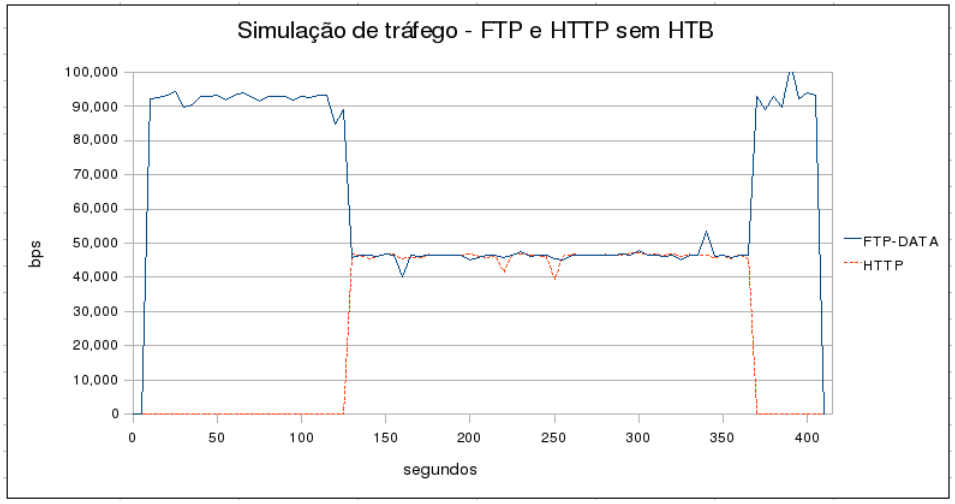

Figura 4.8: Teste 2 - Tráfego sem QoS

Os comandos utilizados para a geração de tráfego deste teste são os mesmos do teste anterior. Podem ser consultados na figura 4.5.

[3] 1500 + 150 + 1500 blocos

Capítulo 5

Conclusão

Conforme mencionado por (LEAL, 2004), as funcionalidades de QoS podem melhorar a percepção do usuário quanto à eficiência dos serviços de comunicação.

Pelo estudo desenvolvido para elaborar o presente trabalho, é possível concluir que o HTB é uma boa ferramenta de QoS e executa com competência seu trabalho. Sua configuração, apesar de ser em interface de texto, é bem muito simples.

Nos testes realizados, a administração e priorização do tráfego pôde ser observada com clareza. Comparando o tráfego do teste 1 com o do teste 2 pode facilmente observar que o usuário que demanda o tráfego HTTP terá uma resposta mais rápida da sua solicitação com o QoS do que sem o QoS, aumentando assim seu QoE ou grau de safisfação com o serviço. A duração do fluxo demonstra isto. Com HTB o tráfego HTTP demorou 135 segundos. Sem HTB o mesmo tráfego demorou 240 segundos.

Em situações reais de produção, o tráfego de correio eletrônico, de FTP ou outra solução de transferência deve ser colocado com prioridade menor (identificador maior) que os tráfegos de aplicações interativas, tipo HTTP ou HTTPS.

Em instalações que utilizam SSH, é necessário cautela com a priorização deste protocolo porque, se o servidor estiver habilitado para fazer transferências de arquivo via SSH, usando SFTP ou SCP haverá perda de performance geral na rede.

Alguns tipos de tráfego são difíceis de identificar e, em ambiente de produção, serão direcionados para a classe *default* e posteriormente, se necessário, devem

ser ajustados. Aplicações sem padrão de porta de comunicação, tais como *msn messenger*, se permitidas na rede, também serão direcionados para a classe *default*.

No ambiente da empresa empregadora do autor, o tráfego de compartilhamento de *drives*, de *Connect-Direct*[1] e de STCP[2] estão colocados com a menor prioridade. A seguir, com média prioridade está o tráfego do correio eletrônico e de HTTP para intranet. Na maior prioridade, tráfego de acesso a aplicações, incluindo Banco de Dados Oracle (porta 1521), *telnet* (porta 23), *ssh* (porta 22) e servidor de aplicação *JBOSS* (porta 8080).

A implantação deste QoS com HTB trouxe uma economia direta para a empresa de aproximadamente cento e setenta e seis mil reais anuais[3]. Não foram consideradas as reduções de custos com administração, porque os links de dados e roteadores não foram instalados, mas o servidor Linux com HTB foi instalado.

No ambiente da empresa, o beneficio do QoS não foi percebido pelos usuários do Ponto Eletrônico, porque esta aplicação foi implementada depois da ativação do HTB. Outras aplicações que já existiam, principalmente as solicitações de autorização de compra e de extratos de cartão, tiveram melhora visível de tempo de resposta e consequente diminuição da quantidade de reclamações dos clientes, pois estas "transações" ganharam uma fatia do link entre São Paulo e Belo Horizonte para si (reserva).

Finalmente, os servidores Linux e Solaris tiveram o acesso à interface gráfica (*X11Forwarding*) e transferência de arquivos através de *ssh* desabilitados.

5.1 Trabalhos Futuros

Outras abordagens de QoS podem ser citadas além do HTB, para uso em situações diversas. Em ambientes menores ou menos complexos, pode ser mais viável a utilização de outras disciplinas de fila *classless*.

Um trabalho posterior de HTB pode abordar as soluções de *front end* para o mesmo. Uma das ferramentas atualmente disponíveis é a **HTB-Tools**[4]. O **HTB-**

[1] Connect-direct é uma solução para transferência de arquivos multi-plataforma da Sterling Commerce, cujo endereço eletrônico é http://www.sterlingcommerce.com.br/

[2] STCP é uma solução para transferência de arquivos multi-plataforma da Riversoft, cujo endereço eletrônico é http://www.riversoft.com.br

[3] Três mil reais/mês de despesa para cada um dos cinco links de dados de 512kbps projetados, menos o valor da subscrição do Red Hat Linux Standard, de R$3.753,00 reais por ano

[4] disponível em http://htb-tools.skydevel.ro/news.php.

Tools esconde a relativa complexidade na criação de classes e posterior marcação de pacotes.

Outras duas opções disponíveis são o **KHTB**, um *front end* gráfico que roda no ambiente Linux e um módulo para **Webmin**[5]. Estas duas ferramentas dependem do script **htb.init**[6].

Uma avaliação comparativa entre as vária disciplinas de fila do Linux, incluindo comparações com soluções livres de outras plataformas, tais como Dummynet ou ALTQ tambem é bastante viável.

[5]Webmin é uma interface de administração web para Linux, Solaris e outros, disponível para *download* em http://www.webmin.com

[6]O htb.init está disponível para download em http://sourceforge.net/projects/htbinit/

Capítulo 6

Referências Bibliográficas

ALMQUIST, P. *RFC1349 – Type of Service in the Internet Protocol Suite [on-line]*. Network Working Group, 1992. Disponível em: <http://www.rfc-editor.org/rfc/rfc1349.txt>.

BALLIACHE, L. *Differentiated Service on Linux HOWTO [on-line]*. [s.n.], 2003. Disponível em: <http://www.opalsoft.net/qos/DS.htm>.

BROWN, M. A. *Traffic Control HOWTO [on-line]*. The Linux Documentation Project, 2006. Disponível em: <http://www.tldp.org/HOWTO/Traffic-Control-HOWTO/index.html>.

DEVERA, M. *HTB Home [on-line]*. [s.n.], 2003. Disponível em: <http://luxik-.cdi.cz/\~devik/qos/htb/>.

HUBERT, B. *Linux Advanced Routing & Traffic Control [on-line]*. Netherlabs BV, 2004. Disponível em: <http://lartc.org/howto/lartc.qdisc.classful.html>.

LEAL, M. A. de A. *QoS – QUALIDADE DE SERVIÇO EM TCP/IP*. 1. ed. Labras: UFLA, 2004.

RECHERT, K. *HFSC Scheduling with Linux [on-line]*. [s.n.], 2005. Disponível em: <http://linux-ip.net/articles/hfsc.en/>.

Apêndice A

Documentos completos

A.1 scripts completos

Code A.1: Script **criabr.sh**

```
1  #!/bin/sh
2  #Script criabr.sh
3  #
4  #Script para criacao de bridge para HTB
5  #Este script nao recebe parametros
6
7  brctl delbr br0
8
9  brctl addbr br0
10 brctl addif br0 eth0
11 brctl addif br0 eth1
12
13 ifconfig br0 192.168.200.201/24 up
```

Code A.2: Script **criafilas.sh**

```
1  #!/bin/sh
2  #Script criafilas.sh
3  #
4  #Script para criacao de filas e classes HTB
5  #Este script nao recebe parametros
6
7  #Limpa as filas previamente criadas
8  tc qdisc del dev eth0 root
9  tc qdisc del dev eth1 root
10
```

```
11
12 ###### Cria classe eth0 -> eth1 (rede interna -> rede externa)↘
   → #######
13
14 tc qdisc add dev eth1 root handle 10: htb default 0
15
16 ###### Cria subclasse por site #######
17
18 ## Recife ##
19 tc class add dev eth1 parent 10: classid 10:2 htb rate 1024↘
   →kbit
20 tc class add dev eth1 parent 10:2 classid 10:21 htb rate 600↘
   →kbit ceil 1024kbit prio 1
21 tc class add dev eth1 parent 10:2 classid 10:22 htb rate 424↘
   →kbit ceil 1024kbit prio 2
22
23 ## Belo Horizonte ##
24 tc class add dev eth1 parent 10: classid 10:3 htb rate 1024↘
   →kbit
25 tc class add dev eth1 parent 10:3 classid 10:31 htb rate 256↘
   →kbit ceil 1024kbit prio 1
26 tc class add dev eth1 parent 10:3 classid 10:32 htb rate 768↘
   →kbit ceil 1024kbit prio 2
27
28 ## Rio de Janeiro ##
29 tc class add dev eth1 parent 10: classid 10:4 htb rate 1536↘
   →kbit
30 tc class add dev eth1 parent 10:4 classid 10:41 htb rate 640↘
   →kbit ceil 1536kbit prio 1
31 tc class add dev eth1 parent 10:4 classid 10:42 htb rate 512↘
   →kbit ceil 1280kbit prio 2
32
33 ## Alphaville ##
34 tc class add dev eth1 parent 10: classid 10:5 htb rate 2048↘
   →kbit
35 tc class add dev eth1 parent 10:5 classid 10:51 htb rate 1024↘
   →kbit ceil 2048kbit prio 1
36 tc class add dev eth1 parent 10:5 classid 10:52 htb rate 1024↘
   →kbit ceil 2048kbit prio 2
37
38 ## Curitiba  ##
39 tc class add dev eth1 parent 10: classid 10:6 htb rate 512kbit
40 tc class add dev eth1 parent 10:6 classid 10:61 htb rate 448↘
   →kbit ceil 512kbit prio 1
41 tc class add dev eth1 parent 10:6 classid 10:62 htb rate 64↘
   →kbit ceil 256kbit prio 2
42
43
```

```
44 ###### Cria classe eth1 -> eth0 (rede externa -> rede interna)↘
   → ###########
45
46 tc qdisc add dev eth0 root handle 11: htb default 0
47
48 ###### Cria subclasse por site #######
49
50 ## Recife ##
51 tc class add dev eth0 parent 11: classid 11:2 htb rate 1024↘
   →kbit
52 tc class add dev eth0 parent 11:2 classid 11:21 htb rate 600↘
   →kbit ceil 1024kbit prio 1
53 tc class add dev eth0 parent 11:2 classid 11:22 htb rate 424↘
   →kbit ceil 1024kbit prio 2
54
55 ## Belo Horizonte ##
56 tc class add dev eth0 parent 11: classid 11:3 htb rate 1024↘
   →kbit
57 tc class add dev eth0 parent 11:3 classid 11:31 htb rate 256↘
   →kbit ceil 1024kbit prio 1
58 tc class add dev eth0 parent 11:3 classid 11:32 htb rate 768↘
   →kbit ceil 1024kbit prio 2
59
60 ## Rio de Janeiro ##
61 tc class add dev eth0 parent 11: classid 11:4 htb rate 1536↘
   →kbit
62 tc class add dev eth0 parent 11:4 classid 11:41 htb rate 640↘
   →kbit ceil 1536kbit prio 1
63 tc class add dev eth0 parent 11:4 classid 11:42 htb rate 512↘
   →kbit ceil 1280kbit prio 2
64
65 ## Alphaville ##
66 tc class add dev eth0 parent 11: classid 11:5 htb rate 2048↘
   →kbit
67 tc class add dev eth0 parent 11:5 classid 11:51 htb rate 1024↘
   →kbit ceil 2048kbit prio 1
68 tc class add dev eth0 parent 11:5 classid 11:52 htb rate 1024↘
   →kbit ceil 2048kbit prio 2
69
70 ## Curitiba  ##
71 tc class add dev eth0 parent 11: classid 11:6 htb rate 512kbit
72 tc class add dev eth0 parent 11:6 classid 11:61 htb rate 448↘
   →kbit ceil 512kbit prio 1
73 tc class add dev eth0 parent 11:6 classid 11:62 htb rate 64↘
   →kbit ceil 256kbit prio 2
```

Code A.3: Script **mrcacao.sh**

```
1 #!/bin/sh
```

```
#Script marcacao.sh

#Limpa tabels mangle do iptables
iptables -t mangle -F


##### MARCA PACOTES #####

##### Recife ############

iptables -t mangle -A POSTROUTING -p tcp -s 10.181.0.0/21 -d 191.167.4.25 -j CLASSIFY --set-class 11:21
iptables -t mangle -A POSTROUTING -p tcp -s 191.167.4.25 -d 10.181.0.0/21 -j CLASSIFY --set-class 10:21

iptables -t mangle -A POSTROUTING -p all -s 10.181.0.0/21 -d ! 191.167.4.25 -j CLASSIFY --set-class 11:22
iptables -t mangle -A POSTROUTING -p all -s ! 191.167.4.25 -d 10.181.0.0/21 -j CLASSIFY --set-class 10:22

##### Belo Horizonte ####

iptables -t mangle -A POSTROUTING -p tcp -s 191.168.0.0/16 -d 191.167.4.25 -j CLASSIFY --set-class 11:31
iptables -t mangle -A POSTROUTING -p tcp -s 191.167.4.25 -d 191.168.0.0/16 -j CLASSIFY --set-class 10:31

iptables -t mangle -A POSTROUTING -p all -s 191.168.0.0/16 -d ! 191.167.4.25 -j CLASSIFY --set-class 11:32
iptables -t mangle -A POSTROUTING -p all -s ! 191.167.4.25 -d 191.168.0.0/16 -j CLASSIFY --set-class 10:32

##### Rio de Janeiro ####

iptables -t mangle -A POSTROUTING -p tcp -s 192.168.0.0/16 -d 191.167.4.25 -j CLASSIFY --set-class 11:41
iptables -t mangle -A POSTROUTING -p tcp -s 191.167.4.25 -d 192.168.0.0/16 -j CLASSIFY --set-class 10:41

iptables -t mangle -A POSTROUTING -p all -s 192.168.0.0/16 -d ! 191.167.4.25 -j CLASSIFY --set-class 11:42
iptables -t mangle -A POSTROUTING -p all -s ! 191.167.4.25 -d 192.168.0.0/16 -j CLASSIFY --set-class 10:42

##### Alphaville ########

iptables -t mangle -A POSTROUTING -p tcp -s 10.170.168.0/21 -d 191.167.4.25 -j CLASSIFY --set-class 11:51
```

```
37 iptables -t mangle -A POSTROUTING -p tcp -s 191.167.4.25 -d ↘
   →10.170.168.0/21 -j CLASSIFY --set-class 10:51
38
39 iptables -t mangle -A POSTROUTING -p all -s 10.170.168.0/21 -d↘
   → ! 191.167.4.25 -j CLASSIFY --set-class 11:52
40 iptables -t mangle -A POSTROUTING -p all -s ! 191.167.4.25 -d ↘
   →10.170.168.0/21 -j CLASSIFY --set-class 10:52
41
42 ##### Curitiba  #########
43
44 iptables -t mangle -A POSTROUTING -p tcp -s 10.141.0.0/16 -d ↘
   →191.167.4.25 -j CLASSIFY --set-class 11:61
45 iptables -t mangle -A POSTROUTING -p tcp -s 191.167.4.25 -d ↘
   →10.141.0.0/16 -j CLASSIFY --set-class 10:61
46
47 iptables -t mangle -A POSTROUTING -p all -s 10.141.0.0/16 -d !↘
   → 191.167.4.25 -j CLASSIFY --set-class 11:62
48 iptables -t mangle -A POSTROUTING -p all -s ! 191.167.4.25 -d ↘
   →10.141.0.0/16 -j CLASSIFY --set-class 10:62
49
50
51 ##### F I M #############
```

A.2 Saída completa dos comandos

Code A.4: Saída do comando **tc class show dev eth1**

```
1  # tc class show dev eth1
2  class htb 10:11 parent 10:1 prio 1 rate 600000bit ceil 1536↘
   →Kbit burst 1674b cburst 1791b
3  class htb 10:22 parent 10:2 prio 2 rate 424000bit ceil 1024↘
   →Kbit burst 1652b cburst 1728b
4  class htb 10:1 root rate 1536Kbit ceil 1536Kbit burst 1791b ↘
   →cburst 1791b
5  class htb 10:32 parent 10:3 prio 2 rate 768000bit ceil 1024↘
   →Kbit burst 1695b cburst 1728b
6  class htb 10:2 root rate 1024Kbit ceil 1024Kbit burst 1728b ↘
   →cburst 1728b
7  class htb 10:31 parent 10:3 prio 1 rate 256000bit ceil 1024↘
   →Kbit burst 1632b cburst 1728b
8  class htb 10:12 parent 10:1 prio 2 rate 936000bit ceil 1536↘
   →Kbit burst 1716b cburst 1791b
9  class htb 10:21 parent 10:2 prio 1 rate 600000bit ceil 1024↘
   →Kbit burst 1674b cburst 1728b
10 class htb 10:3 root rate 1024Kbit ceil 1024Kbit burst 1728b ↘
   →cburst 1728b
```

```
11  class htb 10:4 root rate 1536Kbit ceil 1536Kbit burst 1791b ↘
    →cburst 1791b
12  class htb 10:51 parent 10:5 prio 1 rate 1024Kbit ceil 2048Kbit↘
    → burst 1728b cburst 1856b
13  class htb 10:62 parent 10:6 prio 2 rate 64000bit ceil 256000↘
    →bit burst 1608b cburst 1632b
14  class htb 10:41 parent 10:4 prio 1 rate 640000bit ceil 1536↘
    →Kbit burst 1680b cburst 1791b
15  class htb 10:5 root rate 2048Kbit ceil 2048Kbit burst 1856b ↘
    →cburst 1856b
16  class htb 10:42 parent 10:4 prio 2 rate 512000bit ceil 1280↘
    →Kbit burst 1664b cburst 1760b
17  class htb 10:6 root rate 512000bit ceil 512000bit burst 1664b ↘
    →cburst 1664b
18  class htb 10:52 parent 10:5 prio 2 rate 1024Kbit ceil 2048Kbit↘
    → burst 1728b cburst 1856b
19  class htb 10:61 parent 10:6 prio 1 rate 448000bit ceil 512000↘
    →bit burst 1655b cburst 1664b
```

Code A.5: Saída do comando **tc -s class show dev eth1**

```
1   # tc -s  class show dev eth1
2   class htb 10:11 parent 10:1 prio 1 rate 600000bit ceil 1536↘
    →Kbit burst 1674b cburst 1791b
3    Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
4    rate 0bit 0pps backlog 0b 0p requeues 0
5    lended: 0 borrowed: 0 giants: 0
6    tokens: 22333 ctokens: 9333
7
8   class htb 10:22 parent 10:2 prio 2 rate 424000bit ceil 1024↘
    →Kbit burst 1652b cburst 1728b
9    Sent 7902548869 bytes 9439890 pkt (dropped 0, overlimits 0 ↘
     →requeues 0)
10   rate 0bit 0pps backlog 0b 0p requeues 0
11   lended: 6307872 borrowed: 3132018 giants: 0
12   tokens: 9398 ctokens: 13125
13
14  class htb 10:1 root rate 1536Kbit ceil 1536Kbit burst 1791b ↘
    →cburst 1791b
15   Sent 12814358558 bytes 136124333 pkt (dropped 0, overlimits 0↘
     → requeues 0)
16   rate 3568bit 3pps backlog 0b 0p requeues 0
17   lended: 19064917 borrowed: 0 giants: 0
18   tokens: -1088 ctokens: -1088
19
20  class htb 10:32 parent 10:3 prio 2 rate 768000bit ceil 1024↘
    →Kbit burst 1695b cburst 1728b
21   Sent 5504041553 bytes 20542695 pkt (dropped 0, overlimits 0 ↘
     →requeues 0)
```

```
 rate 2680bit 2pps backlog 0b 0p requeues 0
 lended: 20003404 borrowed: 539291 giants: 403
 tokens: 16500 ctokens: 12625

class htb 10:2 root rate 1024Kbit ceil 1024Kbit burst 1728b ↘
→cburst 1728b
 Sent 7923932079 bytes 9651181 pkt (dropped 0, overlimits 0 ↘
 →requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 3163616 borrowed: 0 giants: 0
 tokens: 13125 ctokens: 13125

class htb 10:31 parent 10:3 prio 1 rate 256000bit ceil 1024↘
→Kbit burst 1632b cburst 1728b
 Sent 28338 bytes 373 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 342 borrowed: 31 giants: 0
 tokens: 48750 ctokens: 12938

class htb 10:12 parent 10:1 prio 2 rate 936000bit ceil 1536↘
→Kbit burst 1716b cburst 1791b
 Sent 12814358471 bytes 136124332 pkt (dropped 0, overlimits 0↘
 → requeues 0)
 rate 3448bit 3pps backlog 0b 0p requeues 0
 lended: 117059416 borrowed: 19064917 giants: 0
 tokens: -9671 ctokens: -1088

class htb 10:21 parent 10:2 prio 1 rate 600000bit ceil 1024↘
→Kbit burst 1674b cburst 1728b
 Sent 21383210 bytes 211291 pkt (dropped 0, overlimits 0 ↘
 →requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 179693 borrowed: 31598 giants: 0
 tokens: -22273 ctokens: -11624

class htb 10:3 root rate 1024Kbit ceil 1024Kbit burst 1728b ↘
→cburst 1728b
 Sent 5504069891 bytes 20543068 pkt (dropped 0, overlimits 0 ↘
 →requeues 0)
 rate 3032bit 2pps backlog 0b 0p requeues 0
 lended: 539322 borrowed: 0 giants: 456
 tokens: 12625 ctokens: 12625

class htb 10:4 root rate 1536Kbit ceil 1536Kbit burst 1791b ↘
→cburst 1791b
 Sent 96451417270 bytes 172764816 pkt (dropped 0, overlimits 0↘
 → requeues 0)
 rate 7464bit 3pps backlog 0b 0p requeues 0
```

```
 lended: 49949512 borrowed: 0 giants: 0
 tokens: 9083 ctokens: 9083

class htb 10:51 parent 10:5 prio 1 rate 1024Kbit ceil 2048Kbit burst 1728b cburst 1856b
 Sent 828814 bytes 3857 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 3241 borrowed: 616 giants: 0
 tokens: -6529 ctokens: -6872

class htb 10:62 parent 10:6 prio 2 rate 64000bit ceil 256000bit burst 1608b cburst 1632b
 Sent 1855505 bytes 32287 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 15526 borrowed: 16761 giants: 0
 tokens: -206480 ctokens: -46730

class htb 10:41 parent 10:4 prio 1 rate 640000bit ceil 1536Kbit burst 1680b cburst 1791b
 Sent 5833209 bytes 17511 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 11916 borrowed: 5595 giants: 0
 tokens: -7359 ctokens: -3276

class htb 10:5 root rate 2048Kbit ceil 2048Kbit burst 1856b cburst 1856b
 Sent 30086504334 bytes 48404695 pkt (dropped 0, overlimits 0 requeues 0)
 rate 2712bit 0pps backlog 0b 0p requeues 0
 lended: 3012556 borrowed: 0 giants: 0
 tokens: 7063 ctokens: 7063

class htb 10:42 parent 10:4 prio 2 rate 512000bit ceil 1280Kbit burst 1664b cburst 1760b
 Sent 96445584061 bytes 172747305 pkt (dropped 0, overlimits 0 requeues 0)
 rate 7496bit 3pps backlog 0b 0p requeues 0
 lended: 122803388 borrowed: 49943917 giants: 0
 tokens: 25250 ctokens: 10700

class htb 10:6 root rate 512000bit ceil 512000bit burst 1664b cburst 1664b
 Sent 1855505 bytes 32287 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
```

```
 lended: 16761 borrowed: 0 giants: 0
 tokens: -16605 ctokens: -16605

class htb 10:52 parent 10:5 prio 2 rate 1024Kbit ceil 2048Kbit burst 1728b cburst 1856b
 Sent 30085675520 bytes 48400838 pkt (dropped 0, overlimits 0 requeues 0)
 rate 6944bit 1pps backlog 0b 0p requeues 0
 lended: 45388898 borrowed: 3011940 giants: 0
 tokens: 13125 ctokens: 7063

class htb 10:61 parent 10:6 prio 1 rate 448000bit ceil 512000bit burst 1655b cburst 1664b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 29571 ctokens: 26000
```

Code A.6: Saída do comando **tc -s -d class show dev eth1**

```
# tc -s -d class show dev eth1
class htb 10:11 parent 10:1 prio 1 quantum 7500 rate 600000bit ceil 1536Kbit burst 1674b/8 mpu 0b overhead 0b cburst 1791b/8 mpu 0b overhead 0b level 0
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 22333 ctokens: 9333

class htb 10:22 parent 10:2 prio 2 quantum 5300 rate 424000bit ceil 1024Kbit burst 1652b/8 mpu 0b overhead 0b cburst 1728b/8 mpu 0b overhead 0b level 0
 Sent 7902548869 bytes 9439890 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 6307872 borrowed: 3132018 giants: 0
 tokens: 9398 ctokens: 13125

class htb 10:1 root rate 1536Kbit ceil 1536Kbit burst 1791b/8 mpu 0b overhead 0b cburst 1791b/8 mpu 0b overhead 0b level 7
 Sent 12814358558 bytes 136124333 pkt (dropped 0, overlimits 0 requeues 0)
 rate 8bit 0pps backlog 0b 0p requeues 0
 lended: 19064917 borrowed: 0 giants: 0
 tokens: -1088 ctokens: -1088

class htb 10:32 parent 10:3 prio 2 quantum 9600 rate 768000bit ceil 1024Kbit burst 1695b/8 mpu 0b overhead 0b cburst 1728b/8 mpu 0b overhead 0b level 0
```

```
 Sent 5504077315 bytes 20542938 pkt (dropped 0, overlimits 0 requeues 0)
 rate 808bit 1pps backlog 0b 0p requeues 0
 lended: 20003646 borrowed: 539292 giants: 407
 tokens: 16500 ctokens: 12625

class htb 10:2 root rate 1024Kbit ceil 1024Kbit burst 1728b/8 mpu 0b overhead 0b cburst 1728b/8 mpu 0b overhead 0b level 7
 Sent 7923932079 bytes 9651181 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 3163616 borrowed: 0 giants: 0
 tokens: 13125 ctokens: 13125

class htb 10:31 parent 10:3 prio 1 quantum 3200 rate 256000bit ceil 1024Kbit burst 1632b/8 mpu 0b overhead 0b cburst 1728b/8 mpu 0b overhead 0b level 0
 Sent 28338 bytes 373 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 342 borrowed: 31 giants: 0
 tokens: 48750 ctokens: 12938

class htb 10:12 parent 10:1 prio 2 quantum 11700 rate 936000bit ceil 1536Kbit burst 1716b/8 mpu 0b overhead 0b cburst 1791b/8 mpu 0b overhead 0b level 0
 Sent 12814358471 bytes 136124332 pkt (dropped 0, overlimits 0 requeues 0)
 rate 8bit 0pps backlog 0b 0p requeues 0
 lended: 117059416 borrowed: 19064917 giants: 0
 tokens: -9671 ctokens: -1088

class htb 10:21 parent 10:2 prio 1 quantum 7500 rate 600000bit ceil 1024Kbit burst 1674b/8 mpu 0b overhead 0b cburst 1728b/8 mpu 0b overhead 0b level 0
 Sent 21383210 bytes 211291 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 179693 borrowed: 31598 giants: 0
 tokens: -22273 ctokens: -11624

class htb 10:3 root rate 1024Kbit ceil 1024Kbit burst 1728b/8 mpu 0b overhead 0b cburst 1728b/8 mpu 0b overhead 0b level 7
 Sent 5504105653 bytes 20543311 pkt (dropped 0, overlimits 0 requeues 0)
 rate 720bit 0pps backlog 0b 0p requeues 0
 lended: 539323 borrowed: 0 giants: 460
 tokens: 12625 ctokens: 12625

```

```
56  class htb 10:4 root rate 1536Kbit ceil 1536Kbit burst 1791b/8 ↘
    →mpu 0b overhead 0b cburst 1791b/8 mpu 0b overhead 0b level 7
57   Sent 96451540748 bytes 172765312 pkt (dropped 0, overlimits 0↘
     → requeues 0)
58   rate 7048bit 3pps backlog 0b 0p requeues 0
59   lended: 49949512 borrowed: 0 giants: 0
60   tokens: 6375 ctokens: 6375
61
62  class htb 10:51 parent 10:5 prio 1 quantum 12800 rate 1024Kbit↘
    → ceil 2048Kbit burst 1728b/8 mpu 0b overhead 0b cburst 1856b↘
    →/8 mpu 0b overhead 0b level 0
63   Sent 828814 bytes 3857 pkt (dropped 0, overlimits 0 requeues ↘
     →0)
64   rate 0bit 0pps backlog 0b 0p requeues 0
65   lended: 3241 borrowed: 616 giants: 0
66   tokens: -6529 ctokens: -6872
67
68  class htb 10:62 parent 10:6 prio 2 quantum 1000 rate 64000bit ↘
    →ceil 256000bit burst 1608b/8 mpu 0b overhead 0b cburst 1632b↘
    →/8 mpu 0b overhead 0b level 0
69   Sent 1855505 bytes 32287 pkt (dropped 0, overlimits 0 ↘
     →requeues 0)
70   rate 0bit 0pps backlog 0b 0p requeues 0
71   lended: 15526 borrowed: 16761 giants: 0
72   tokens: -206480 ctokens: -46730
73
74  class htb 10:41 parent 10:4 prio 1 quantum 8000 rate 640000bit↘
    → ceil 1536Kbit burst 1680b/8 mpu 0b overhead 0b cburst 1791b↘
    →/8 mpu 0b overhead 0b level 0
75   Sent 5833209 bytes 17511 pkt (dropped 0, overlimits 0 ↘
     →requeues 0)
76   rate 0bit 0pps backlog 0b 0p requeues 0
77   lended: 11916 borrowed: 5595 giants: 0
78   tokens: -7359 ctokens: -3276
79
80  class htb 10:5 root rate 2048Kbit ceil 2048Kbit burst 1856b/8 ↘
    →mpu 0b overhead 0b cburst 1856b/8 mpu 0b overhead 0b level 7
81   Sent 30086566204 bytes 48404790 pkt (dropped 0, overlimits 0 ↘
     →requeues 0)
82   rate 264bit 0pps backlog 0b 0p requeues 0
83   lended: 3012556 borrowed: 0 giants: 0
84   tokens: 3889 ctokens: 3889
85
86  class htb 10:42 parent 10:4 prio 2 quantum 6400 rate 512000bit↘
    → ceil 1280Kbit burst 1664b/8 mpu 0b overhead 0b cburst 1760b↘
    →/8 mpu 0b overhead 0b level 0
87   Sent 96445707539 bytes 172747801 pkt (dropped 0, overlimits 0↘
     → requeues 0)
```

```
 rate 6544bit 3pps backlog 0b 0p requeues 0
 lended: 122803884 borrowed: 49943917 giants: 0
 tokens: 17125 ctokens: 7450

class htb 10:6 root rate 512000bit ceil 512000bit burst 1664b/8 mpu 0b overhead 0b cburst 1664b/8 mpu 0b overhead 0b level 7
 Sent 1855505 bytes 32287 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 16761 borrowed: 0 giants: 0
 tokens: -16605 ctokens: -16605

class htb 10:52 parent 10:5 prio 2 quantum 12800 rate 1024Kbit ceil 2048Kbit burst 1728b/8 mpu 0b overhead 0b cburst 1856b/8 mpu 0b overhead 0b level 0
 Sent 30085737390 bytes 48400933 pkt (dropped 0, overlimits 0 requeues 0)
 rate 256bit 0pps backlog 0b 0p requeues 0
 lended: 45388993 borrowed: 3011940 giants: 0
 tokens: 6764 ctokens: 3889

class htb 10:61 parent 10:6 prio 1 quantum 5600 rate 448000bit ceil 512000bit burst 1655b/8 mpu 0b overhead 0b cburst 1664b/8 mpu 0b overhead 0b level 0
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 29571 ctokens: 26000
```

Code A.7: Saída do comando **iptables -L -t mangle -v**

```
# iptables -L -t mangle -v
Chain PREROUTING (policy ACCEPT 74G packets, 11T bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain INPUT (policy ACCEPT 719K packets, 65M bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 74G packets, 11T bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 267K packets, 32M bytes)
 pkts bytes target     prot opt in     out     source               destination

```

```
Chain POSTROUTING (policy ACCEPT 74G packets, 11T bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 CLASSIFY   tcp  --  any    any     191.165.0.0/16       191.167.4.25         CLASSIFY set 11:11
    0     0 CLASSIFY   tcp  --  any    any     191.167.4.25         191.165.0.0/16       CLASSIFY set 10:11
    0     0 CLASSIFY   all  --  any    any     191.165.0.0/16      !191.167.4.25         CLASSIFY set 11:12
 317M   29G CLASSIFY   all  --  any    any    !191.167.4.25         191.165.0.0/16       CLASSIFY set 10:12
 143M   22G CLASSIFY   tcp  --  any    any     10.181.0.0/21        191.167.4.25         CLASSIFY set 11:21
 116M   43G CLASSIFY   tcp  --  any    any     191.167.4.25         10.181.0.0/21        CLASSIFY set 10:21
 294M   72G CLASSIFY   all  --  any    any     10.181.0.0/21       !191.167.4.25         CLASSIFY set 11:22
 255M  248G CLASSIFY   all  --  any    any    !191.167.4.25         10.181.0.0/21        CLASSIFY set 10:22
2782K  159M CLASSIFY   tcp  --  any    any     191.168.0.0/16       191.167.4.25         CLASSIFY set 11:31
2598K  180M CLASSIFY   tcp  --  any    any     191.167.4.25         191.168.0.0/16       CLASSIFY set 10:31
 989M  422G CLASSIFY   all  --  any    any     191.168.0.0/16      !191.167.4.25         CLASSIFY set 11:32
1088M  200G CLASSIFY   all  --  any    any    !191.167.4.25         191.168.0.0/16       CLASSIFY set 10:32
 130M   22G CLASSIFY   tcp  --  any    any     192.168.0.0/16       191.167.4.25         CLASSIFY set 11:41
 108M   47G CLASSIFY   tcp  --  any    any     191.167.4.25         192.168.0.0/16       CLASSIFY set 10:41
5642M  845G CLASSIFY   all  --  any    any     192.168.0.0/16      !191.167.4.25         CLASSIFY set 11:42
1363M 1004G CLASSIFY   all  --  any    any    !191.167.4.25         192.168.0.0/16       CLASSIFY set 10:42
  70M   13G CLASSIFY   tcp  --  any    any     10.170.168.0/21      191.167.4.25         CLASSIFY set 11:51
  57M   21G CLASSIFY   tcp  --  any    any     191.167.4.25         10.170.168.0/21      CLASSIFY set 10:51
1020M  305G CLASSIFY   all  --  any    any     10.170.168.0/21     !191.167.4.25         CLASSIFY set 11:52
1037M  597G CLASSIFY   all  --  any    any    !191.167.4.25         10.170.168.0/21      CLASSIFY set 10:52
 598K   96M CLASSIFY   tcp  --  any    any     10.141.0.0/16        191.167.4.25         CLASSIFY set 11:61
 439K  194M CLASSIFY   tcp  --  any    any     191.167.4.25         10.141.0.0/16        CLASSIFY set 10:61
```

```
38  3632K  933M CLASSIFY  all  --  any  any  10.141.0.0/16 ↘
    →  !191.167.4.25  CLASSIFY set 11:62
39  4531K 1941M CLASSIFY  all  --  any  any  !191.167.4.25 ↘
    →  10.141.0.0/16  CLASSIFY set 10:62
```

A.3 Utilização dos programas de teste

O **socketserver** aceita o parâmetro "-h", que solicita a ajuda do programa. Também aceita os parâmetros opcionais "-v", "-vv", "-b xxx" e "-w <nome>", respectivamente para mostrar algumas informações do funcionamento (*verbosity*), mostrar muitas informações do funcionamento, alterar o tamanho do bloco de comunicação para "xxx *bytes*" e para gravar o arquivo <nome> com informações das taxas de comunicação de cada *thread*. O tamanho padrão de bloco é de 1024 bytes. Caso o tamanho do bloco seja alterado, deverá ser utilizado o mesmo tamanho no **socketclient**.

O formato do arquivo gravado, quando o usuário passa a opção "-w", é mostrado na figura A.1. O programa **socketserver** sempre abre o arquivo para gravação sem apagar o conteúdo (modo *append*). A primeira linha gravada no arquivo, em cada execução, é um cabeçalho. O formato foi projetado para utilização em um programa de planilha eletrônica tipo o *Open Office Calc* e contém os campos: data padrão Unix[1], a taxa parcial trafegada na primeira porta, a taxa parcial trafegada na segunda porta e assim sucessivamente, uma coluna para cada porta passada como parâmetro. A taxa parcial é calculada pela *thread* após cada recepção (ou transmissão) de um bloco, com precisão de milissegundos. O *overhead* de comunicação (*frame* TCP e *datagrama* IP) não é considerado neste cálculo. Após o término da comunicação de uma *thread*, os valores de estatística desta *thread* é zerado por ela mesma. O log é gravado por uma função acionada por evento de relógio, que ocorre a cada cinco segundos.

O programa **socketclient** deve receber como parâmetros "-s IP" e "-t|-T|-r|-R porta tamanho sleep". Respectivamente o endereço IP do **socketserver** e os parâmetros para cada **thread** transmitir (se -t ou -T) ou receber (se -r ou -R), por qual porta, quantos blocos devem trafegar e, finalmente, quantos segundos a *thread* deve esperar antes de iniciar a comunicação. Opcionalmente podem ser passados os parâmetros "-h", "-v", "-vv" e "-b xxx", respectivamente para consultar a ajuda do programa, para o programa mostrar algumas informações durante o funciona-

[1] Quantidade de segundos decorridos desde 01/01/1970

```
1     data     ,      3001,      3002,      3003,      3004,      3005
2  1250342677,      0.00,      0.00,      0.00,      0.00,      0.00
3  1250342682,      0.00,      0.00,      0.00,      0.00,      0.00
4  1250342687,  31982.03,      0.00,      0.00,      0.00,      0.00
5  1250342692,  32507.56,      0.00,      0.00,      0.00,      0.00
6  1250342697,  32503.80,      0.00,      0.00,      0.00,      0.00
7  1250342702,   6486.76,  45750.76,      0.00,      0.00,      0.00
8  1250342707,  29249.72,  46718.51,      0.00,      0.00,      0.00
9  1250342712,  21114.12,  46565.80,      0.00,      0.00,      0.00
10 1250342717,   4718.33,   4423.88,  31902.07,      0.00,      0.00
11 1250342722,   1475.46,   9375.46,  40835.23,      0.00,      0.00
12 1250342727,   4729.11,   9391.01,  31041.72,      0.00,      0.00
13 1250342732,  30154.25,  19874.09,      0.00,      0.00,      0.00
14 1250342737,  33015.07,  23250.73,      0.00,      0.00,      0.00
```

Figura A.1: Gravação do registro de atividades do programa **socketserver**

mento, para mostrar muitas informações e, finalmente, para especificar o tamanho do bloco.

Foi implementado um protocolo simples entre os programas para a comunicação. Não existe verificação do conteúdo dos dados trafegados, pois o objetivo é somente gerar tráfego. No inicio da sua operação, o programa **socketserver** abre uma *thread* para cada porta especificada como parâmetro e fica escutando-as (*listen mode*). O programa **socketclient**, de acordo com as opções recebidas, conecta-se na porta especificada, no IP do equipamento onde roda o **socketserver**. O **socketclient** envia então uma mensagem com 10 *bytes* contendo a opção de comunicação (transmissão ou recepção) e a quantidade de blocos a serem comunicados. Cada *thread* inicia a transmissão (ou recepção) dos blocos, até a quantidade especificada. Se o usuário solicitar ao **socketclient** que durma no inicio de uma **thread**, esta ficará parada pela quantidade de segundos passada como parâmetro e somente depois inicia o tráfego de dados.

A **thread** que recebe os dados, envia ao transmissor um *byte* logo após receber um bloco, informando que a recepção foi concluída. O transmissor somente inicia o envio de um novo bloco depois que recebe este *byte* de controle (*handshake*). Quando o parâmetro de comunicação do **socketclient** for "-T" ou "-R", a comunicação será feita sem o *handshake*.

Como as *threads* do **socketclient** e do **socketserver** operam de maneira relativamente autônoma, não é necessário que o **socketclient** se conecte a todas as portas disponíveis no **socketserver** a cada execução. Após o término da comunicação de todas as *threads*, o programa **socketclient** termina.

A.4 Listagem dos programas de teste

Code A.8: Fonte do programa **socketclient**

```
/*
Copyright (C) 2009 by Jose Geraldo de Oliveira
jotagera@gmail.com

This program is free software; you can redistribute it and/or ↘
→modify it under the terms of the GNU General Public License ↘
→as published by the Free Software Foundation; either version ↘
→2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful↘
→, but WITHOUT ANY WARRANTY; without even the implied warranty↘
→ of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See↘
→ the GNU General Public License for more details.

You should have received a copy of the GNU General Public ↘
→License along with this program; if not, write to the Free ↘
→Software Foundation, Inc., 59 Temple Place - Suite 330, ↘
→Boston, MA 02111-1307, USA.
*/

/*
* Registro de alteracoes
* Programa socketclient v0.1 de 15/07/2009
* JotaGera
*
* v0.2 de 24/07/2009
* Mudei os parametros de trafego para 1k cada unidade.
* Inclui as opcoes -h, -v e -vv
*
* v0.3 de 25/07/2009
* Inclui opcao de sleep no cliente
* Inclui opcao de tragego sem handshake
*
* v0.4 de 31/07/2009
* Incluida somente para niverar as versoes do cliente e do ↘
→server
*
* v0.5 de 31/07/2009
* Retirei opcao de numero magico 1313
* Inclui pparametro -b para especificar o blocksize. Por ↘
→omissao o
* blocksize e' 1024
*
* v0.6 de 07/08/2009
```

```
* Mudei o sleep programado para depois do connect, para ↘
→melhorar
* o log do lado do server.
*/

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>

struct parm {
   char ip[17];
   char func;
   int port;
   int tama;
   int sleep;
};

void *tcp_client(struct parm *parm);

void perror(const char *s);
int errno,verbose,blocksize;

void help(int exit_code) {

char *msg = " \
Programa para cliente de socket tcp/ip.\n\
Jotagera   -   Ufla 2.009   -   v0.1\n\n\
Uso: socketclient [-h] [-v] [-vv] [-b xxx] -s ip_server -t|-r ↘
→porta1 tamanho1 \\\n\
   sleep1 -t|-r portan tamanhon sleepn\n\
\n\
O parametro -s especifica o ip de destino (o server).\n\
O parametro -b especifica o tamanho do bloco. Por omissao e' ↘
→1024.\n\
O parametro -t seguido de porta tcp, tamanho e sleep, ↘
→especifica que o\n\
programa deve abrir uma conexao nesta porta e transmitir o ↘
→tamanho\n\
```

```
informado, apos dormir (esperar) sleep1 segundos. Da mesma ↘
→maneira se \n\
comporta o parametro -r para recepcao.\n\
O tamanho e' em kbytes ou seja, se passar 10, sera' trafegado ↘
→10kbytes\n\
Os parametros opcionais -v e -vv mostram mensagens sobre o ↘
→comportamento\n\
do programa, durante sua execucao. -vv mostra mais informacoes↘
→ que -v.\n\
Se usar os parametros T ou R em substituicao aos parametros t ↘
→ou r, a \n\
comunicacao sera' efetuada sem handshake de aplicacao.\n\n\
Este programa aceita no maximo 8 conexoes simultaneas.\n\n\
Use -h para exibir este help.\n\n\
Use o programa \"socketserver\" para interagir com este.\n";

   printf ("%s",msg);
   exit (exit_code);

}

main(int argc, char *argv[])
{
   pthread_t thread[8];
   int  iret[8],resp,count,count2,qthreads,port[8],tama[8],↘
   →sleep[8];
   char tmp1[16],func[8],ip[16];
   struct parm parm[8];

   if (argc == 1)
      help(1);

   qthreads = verbose = -1;
   blocksize = 1024;
   ip[0] = '\0';

   for (count = 1; count < argc; count++) {
      if (strncmp(argv[count],"-h",2) == 0)
         help(1);

      if (strncmp(argv[count],"-S",2) == 0 || strncmp(argv[↘
      →count],"-s",2) == 0 ) {
         strncpy(ip,argv[++count],sizeof(ip));
         continue;
      }
      if (strlen(argv[count])>2) {
         if (strncmp(argv[count],"-vv",3) == 0) {
            verbose = 2;
```

```
            continue;
        }
    }
    if (strncmp(argv[count],"-v",2) == 0) {
        verbose = 1;
        continue;
    }
    if (strncmp(argv[count],"-b",2) == 0) { // eh tamanho de↘
    → bloco
        if (count+1 >= argc) {
            puts ("Especifice tamanho do bloco, para -b");
            exit(1);
        }
        blocksize = atoi(argv[count+1]);
        if ( blocksize < 1) {
            puts ("Tamanho de blocksize invalido");
            exit(1);
        }
        count++;
        continue;
    }

    if (strncmp(argv[count],"-t",2) == 0 ||
         strncmp(argv[count],"-T",2) == 0 ||
         strncmp(argv[count],"-r",2) == 0 ||
         strncmp(argv[count],"-R",2) == 0) {
        if (count + 4 > argc) {
            printf ("Faltam valores para parametro -t ou -r. ↘
            →Use '%s -h' para help.\n",argv[0]);
            exit(1);
        }
        qthreads++;
        func[qthreads] = *(argv[count]+1);
        port[qthreads] = atoi(argv[count+1]);
        tama[qthreads] = atoi(argv[count+2]);
        sleep[qthreads] = atoi(argv[count+3]);
        if (port[qthreads] < 1 || port[qthreads] > 65535) {
            printf ("Foi passada uma porta invalida.\n");
            printf ("porta deve ser um numero inteiro entre 1 ↘
            →e 65535.\n");
            exit(1);
        }
        if (tama[qthreads] < 1 || tama[qthreads] > 999999999)↘
        → {
            printf ("Foi passado um tamanho invalido.\n");
            printf ("tamanho deve ser um numero inteiro entre ↘
            →1 e 999999999.\n");
            exit(1);
```

```
            }
            if (sleep[qthreads] < 0) {
               printf ("valor de sleep nao pode ser menor que \
               →zero");
               exit(1);
            }
            for (count2 = 0; count2 < qthreads; count2++) {
               if (port[qthreads] == port[count2]) {
                  printf ("Mesma porta %d passada duas vezes. \
                  →Verifique\n",port[qthreads]);
                  exit(1);
               }
            }
            count+=3;
         }
      }
      if (qthreads == -1 || ip[0] == '\0')  {
         printf ("Faltam parametros para execucao. Use %s -h para\
         → help\n",argv[0]);
         exit(1);
      }

      printf ("Usando blocksize de %d bytes\n",blocksize);

      if (verbose == 2) {
         printf ("Comunicando com %s\n",ip);
         if (qthreads == 0)
            printf ("Rodando com 1 thread com funcao %c, porta %d\
            →, tamanho %d\n",
               func[0],port[0],tama[0]);
         else {
            printf ("Rodando com %d threads nas portas:\n",\
            →qthreads+1);
            for (count = 0; count <= qthreads ; count++)
               printf ("thread %d: funcao %c, porta %d, tamanho %\
               →d\n",
                  count,func[count],port[count],tama[count]);
         }
      }

      for (count = 0; count <= qthreads; count++) {
         strcpy (parm[count].ip,ip);
         parm[count].func  = func[count];
         parm[count].port  = port[count];
         parm[count].tama  = tama[count];
         parm[count].sleep = sleep[count];
         iret[count] =
```

```
            pthread_create(&thread[count],NULL,(void *)tcp_client
            →, (void *) &parm[count]);
            if (iret[count] != 0)
               printf ("Erro %d na criacao da thread #%d\n",iret[
               →count],count);
         if (verbose > 1)
            printf ("Criada thread %d para porta %d\n",count,port
            →[count]);
      }

      fflush(NULL);

      if (verbose == 1)
         printf ( "programa principal vai entrar em wait e
         →aguardar o termino das %d threads\n",qthreads);
      for (count = 0; count <= qthreads; count++) {
         iret[count] = pthread_join( thread[count], NULL);
            if (iret[count] != 0)
               printf ("Erro %d na monitoracao da thread %d\n",
               →iret[count],count);
      }

      exit(0);
   }

   void *tcp_client(struct parm *parm)
   {
      struct sockaddr_in serv_addr;
      int     thread,sockfd,resp,tot,count;
      char    ident[128],message[512],area[15],*buf;
      time_t  ini,now;
      struct tm *tm;

      sleep (1); // thread cliente sempre dorme um segundo, para
      →as mensagens
                 // na tela ficarem bonitinhas
      thread = (int)pthread_self();
      sprintf (ident,"Thread #%d",thread);

      printf ("%s comunicando com %s %c %d %d,sleep de %d\n",
         ident, parm->ip, parm->func, parm->port, parm->tama,
         →parm->sleep);
      if (parm->func == 'T' || parm->func == 'R')
         printf ("%s comunicacao sem handshake\n",ident);
      else
         printf ("%s comunicacao com handshake\n",ident);

      bzero((char *)&serv_addr, sizeof(serv_addr));
```

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr(parm->ip);
serv_addr.sin_port = htons((short int)parm->port);

sockfd = socket (PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sockfd == -1) {
   sprintf (message,"  %s: %s",ident,"create socket error")↘
   →;
   perror (message);
   pthread_exit((void *)1);
}

resp = connect(sockfd,(struct sockaddr *) &serv_addr,sizeof↘
→(serv_addr));
if (resp == -1) {
   sprintf (message,"  %s: %s",ident,"connect error");
   perror (message);
   pthread_exit((void *)1);
}

if (parm->sleep > 0) {
   printf ("%s dormindo %d segundo(s)\n",ident,parm->sleep)↘
   →;
   sleep (parm->sleep);
}

if (parm->func == 't' || parm->func == 'T') { // esta ↘
→thread vai transmitir
   if (parm->func == 't')
      area[0] = 'r';  // o remoto vai receber
   else
      area[0] = 'R';  // o remoto vai receber sem handshake
   sprintf (area+1,"%09d",parm->tama);
   buf = malloc (blocksize);
   if ( buf == NULL ) {
      sprintf (message,"  %s: %s",ident,"malloc error");
      perror (message);
      pthread_exit((void *)1);
   }
   memset ( buf, 'T', blocksize );
   resp = write (sockfd, area, 10);
   if (resp != 10 ) {
      sprintf (message,"  %s: %s",ident,"write header error↘
      →");
      perror (message);
   }
   ini = time(NULL);
   if (verbose >= 1)
```

```
            printf ("%s funcao TX para porta %d, %dkbytes\n",↘
            →ident,parm->port,parm->tama);
         for (count = 0; count < parm->tama; count++) {
            if (verbose == 2)
               printf ("%s funcao TX para porta %d, bloco %d\n",↘
               →ident,parm->port,count);
            resp = write (sockfd, buf, blocksize);
            if (resp != blocksize ) {
               sprintf (message,"  %s: %s",ident,"write buffer ↘
               →error");
               perror (message);
            }
            if (parm->func == 't')
               read (sockfd,buf,1);  // le um byte de sincronismo↘
               → se handshake
         }
      }
      else {
         if (parm->func == 'r')
            area[0] = 't';  // o remoto vai transmitir
         else
            area[0] = 'T';  // o remoto vai transmitir sem ↘
            →handshake
         sprintf (area+1,"%09d",parm->tama);
         buf = malloc (blocksize);
         if ( buf == NULL ) {
            sprintf (message,"  %s: %s",ident,"malloc error");
            perror (message);
            pthread_exit((void *)1);
         }
         resp = write (sockfd, area, 10);
         if (resp != 10 ) {
            sprintf (message,"  %s: %s",ident,"write header error↘
            →");
            perror (message);
         }
         ini = time(NULL);
         if (verbose >= 1)
            printf ("%s funcao RX para porta %d, %dkbytes\n",↘
            →ident,parm->port,parm->tama);
         for (count = 0; count < parm->tama; count++) {
            if (verbose == 2)
               printf ("%s funcao RX para porta %d, bloco %d\n",↘
               →ident,parm->port,count);
            resp = read (sockfd, buf, blocksize);
            if (resp != blocksize ) {
               sprintf (message,"  %s: %s",ident,"read error");
               perror (message);
```

```
            }
            if (parm->func == 'r')
                write (sockfd,"X",1); // grava um byte de sincronismo
        }
    }

    now = time(NULL);

    tot = now-ini;
    if (tot == 0)
        resp=0;
    else
        resp=(parm->tama * blocksize * 8) / tot;

    printf ("%s terminou, %d segundos, %dbps\n",ident,tot,resp);

    close (sockfd);

    pthread_exit((void *)1);
}
```

Code A.9: Fonte do programa **socketserver**

```
/*
Copyright (C) 2009 by Jose Geraldo de Oliveira
jotagera@gmail.com

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
*****************************************************************/


/*
 * Registro de alteracoes
```

```
 * Programa socketserver v0.1 de 15/07/2009
 * JotaGera
 *
 * v0.2 de 24/07/2009
 * Mudei os parametros de trafego para 1k cada unidade.
 * Inclui as opcoes -h, -v e -vv
 *
 * v0.3 de 25/07/2009
 * Inclui opcao de tragego sem handshake
 *
 * v0.4 de 25/07/2009
 * Inclui opcao de gravar estatisticas em arquivo
 *
 * v0.5 de 31/08/2009
 * Retirei a opcao do numero magico 1313 fazer bloco de 1 byte↘
→.
 * Inclui parametro -b nos dois lados, para especificar o ↘
→tamanho do
 *     bloco, em bytes. Por omissao, o tamanho sera' 1024 (1k)
 * Inclui estatistica por bloco, alem da estatistica global ↘
→que ja existia
 *
 * v0.6 de 07/08/2009
 * Alterei a gravacao de log para uma funcao acionada por ↘
→trigger de hora
 * (alarm). Desta maneira, consigo gravar a informacao de cada↘
→ thread
 * em um mesmo instante, melhorando muito para a geracao de ↘
→graficos.
 * As threads alteram uma variavel global para cada, ↘
→informando sua
 * taxa.
 * O log e' gravado em um arquivo somente, cujo nome eh ↘
→informado
 * como marametro da opcao -w.
*/

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <unistd.h>
#include <signal.h>
```

```
#include <time.h>
#include <sys/time.h>

//////////Variaveis globais
struct dparm {
   int id;
   int port;
   double taxap;
   double taxat;
} parm[8];
int qthreads,errno,verbose,gravar,blocksize;
FILE *spool;

//////////Declaracoes de prototipos de funcoes
void sig_handler(int sig);
void *tcp_server(struct dparm *parm);
void perror(const char *s);


void help(int exit_code) {
   puts ("Programa para servidor de socket tcp/ip.");
   puts ("Jotagera   -   Ufla 2.009   -   v0.1\n");

   puts ("Uso: socket_server [-h] [-v] [-vv] [-b xxx] [-w nome\
   →] porta1 porta2 portan\n");

   puts ("Cada parametro passado e' uma porta tcp/ip que o \
   →programa escutara'");
   puts ("atraves de uma thread. Use o socket_client para \
   →interagir com este.\n");
   puts ("Aceita no maximo 8 portas.\n");
   puts ("Os parametros -v e -vv definem niveis de informacao \
   →da operacao");
   puts ("do programa. O parametro -w habilita a gravacao do \
   →log de trafego");
   puts ("no arquivo chamado nome. A informacao sera \
   →acrescentada ao arquivo. ");
   puts ("O parametro -b especifica o tamanho do bloco. Por \
   →omissao e' 1024.");
   puts ("O parametro -h exibe este help");
   exit (exit_code);

}

main(int argc, char *argv[])
{
   pthread_t thread[8];
   int  iret[8],count,count2,len,port[8];
```

```
char filename[64];

if (argc == 1)  {
   help(1);
}

qthreads = verbose = gravar = -1;
blocksize = 1024;

unlink ("-w");

for (count = 1; count < argc; count++) {
   if (*argv[count] == '-') {  // Eh uma opcao
      len=strlen(argv[count]);
      if (len > 1) {  // pode ser -h ou -v
         if (strncmp(argv[count],"-h",2) == 0) // eh help
            help(0);
         if (strncmp(argv[count],"-w",2) == 0) {  // eh ↘
         →para gravar log
            gravar = 1;
            if (count+1 > argc) {
               puts ("Especifique o nome do arquivo a ↘
               →gravar");
               exit(1);
            }
            if (strlen(argv[++count]) > 63) {
               puts ("Nome do arquivo muito longo");
               exit(1);
            }
            strncpy(filename,argv[count],64);
            continue;
         }
         if (strncmp(argv[count],"-v",2) == 0)  // eh ↘
         →verbose 1
            verbose = 1;
         if (strncmp(argv[count],"-b",2) == 0) { // eh ↘
         →tamanho de bloco
            if (count+1 >= argc) {
               puts ("Especifice tamanho do bloco, para -b"↘
               →);
               exit(1);
            }
            blocksize = atoi(argv[count+1]);
            if ( blocksize < 1) {
               puts ("Tamanho de blocksize invalido");
               exit(1);
            }
            count++;
```

```
            continue;
         }
         if (len > 2 && strncmp(argv[count],"-vv",3) == 0) ↘
         → // eh verbose 2
            verbose = 2;
         continue;
      }
      puts ("Opcao invalida. Use -h para help");
      exit (1);
   }
   qthreads++;
   if (qthreads > 7 ){ // de zero a sete
      puts ("Programa so suporta ate' 8 threads.");
      exit (1);
   }
   port[qthreads] = atoi(argv[count]);
   if (port[qthreads] < 1 || port[qthreads] > 65535) {
      printf ("Porta passada no parametro %d e' invalida.\n↘
      →",count+1);
      printf ("porta deve ser um numero inteiro entre 1 e ↘
      →65535.\n");
      printf ("So' root pode 'escutar' em porta menor que ↘
      →1024.\n");
      exit(1);
   }
   for (count2 = 0; count2 < qthreads; count2++) {
      if (port[qthreads] == port[count2]) {
         printf ("porta %d passada duas vezes. corrija\n",↘
         →port[count]);
         exit(1);
      }
   }
}

if (qthreads < 0) {
   puts ("Necessario informar pelo menos uma porta");
   exit (1);
}

if (verbose > 0) {
   if (qthreads == 0)
      printf ("Rodando com 1 thread na porta %d\n",port[0])↘
      →;
   else {
      printf ("Rodando com %d threads nas portas:\n",↘
      →qthreads+1);
      for (count = 0; count <= qthreads ; count++)
```

```
            printf ("thread %d: porta %d\n",count,port[count])↘
            →;
      }
      printf ("Usando blocksize de %d\n",blocksize);
   }

   for (count = 0; count < 8; count++)  // limpa a estrutura
      parm[count].taxap = parm[count].taxat = 0;

   if (gravar == 1) {
      spool = fopen (filename,"a");
      if (spool == NULL) {
         printf ("Erro de abertura do arquivo '%s' para ↘
         →gravacao\n",filename);
         exit (1);
      }
      fprintf (spool,"   data   ");
      for (count = 0; count <= qthreads; count++)
         fprintf (spool,",  %7d",port[count]);
      fprintf (spool,"\n");
   }

   for (count = 0; count <= qthreads; count++) {
         parm[count].port = port[count];
         parm[count].id = count;
         iret[count] =
            pthread_create(&thread[count],NULL,(void *)↘
            →tcp_server,
                           (void *) &parm[count]);
            if (iret[count] != 0)
               printf ("Erro %d na criacao da thread #%d\n",↘
               →iret[count],count);
   }

   fflush(NULL);

   signal (SIGALRM, sig_handler);
   alarm (5);  // seta alarme para cada tres segundos


   for (count = 0; count <= qthreads; count++) {
      iret[count] = pthread_join( thread[count], NULL);
         if (iret[count] != 0)
            printf ("Erro %d pthread_join %d\n",iret[count],↘
            →count);
   }

   exit(0);
```

```
}

void *tcp_server(struct dparm *parm)
{
   struct sockaddr_in server;
   int     thread,sockfd,resp,len_server,sock_connected,ip[4],
   →isize,count;
   char    ident[128],message[512],ip_client[16],func,ssize
   →[10];
   char    *buf;
   time_t now,ini;
   struct tm *tm;
   struct timeval tv;
   struct hostent *hostent;
   double antes,agora,agorinha,tot,bps,bpsparc;


   thread = (int)pthread_self();
   sprintf (ident,"Thread #%d (%d)",thread,parm->id);

   sockfd = socket (PF_INET, SOCK_STREAM, IPPROTO_TCP);
   if (sockfd == -1) {
      sprintf (message,"  %s: %s",ident,"create socket error")
      →;
      perror (message);
      pthread_exit((void *)1);
   }

   server.sin_family = AF_INET;
   server.sin_addr.s_addr = INADDR_ANY; // listen all local
   →address
   server.sin_port = htons((short int)parm->port); // listen
   →to port

   resp = bind (sockfd, (struct sockaddr *)&server, sizeof(
   →server));
   if (resp == -1) {
      sprintf (message,"  %s: %s",ident,"bind error");
      perror (message);
      pthread_exit((void *)1);
   }

   while (1 == 1) {

      printf ("%s inicio do loop. Escutando porta %d\n",ident,
      →parm->port);

      resp = listen(sockfd, 255);
```

```
if (resp == -1) {
   sprintf (message," %s: %s",ident,"listen error");
   perror (message);
   pthread_exit((void *)1);
}

len_server = sizeof(server);
sock_connected = accept(sockfd,(struct sockaddr *) &↘
→server, &len_server);
if (sock_connected == -1) {
   sprintf (message," %s: %s",ident,"accept error");
   perror (message);
   pthread_exit((void *)1);
}

resp = gettimeofday(&tv,NULL);
antes = agorinha = tv.tv_sec + tv.tv_usec / 1000000.0;
ini = tv.tv_sec;
tm = localtime (&ini);
strcpy((char *)ip_client,(char *)inet_ntoa(server.↘
→sin_addr));

sprintf (message,"%s %s",ident,"conectado");
printf ("%s, cliente %s as %02d:%02d:%02d\n",
   message,ip_client,tm->tm_hour,tm->tm_min,tm->tm_sec);

resp = read (sock_connected, &func, 1); // le a funcao ↘
→solicitada
if (resp == -1)
   printf ("Erro de leitura da funcao");
resp = read (sock_connected, ssize, 9); // le o tamanho ↘
→a transmitir
if (resp == -1)
   printf ("Erro de leitura do tamanho");
ssize[9] = '\0'; // finaliza a string
isize = atoi (ssize);
if (verbose > 1)
   printf ("%s recebeu parametros '%c' '%d'\n",ident,↘
   →func,isize);

if (func == 't' || func == 'T') { // e' para este no' ↘
→transmitir
   if (verbose >= 1) {
      printf ("%s funcao TX para porta %d, %dkbytes\n",↘
      →ident,parm->port,isize);
      if (func == 'T')
         printf ("%s comunicando sem handshake, ↘
         →blocksize=%d\n",
```

```
                    ident,blocksize);
            else
                printf ("%s comunicando com handshake, ↘
                →blocksize=%d\n",
                    ident,blocksize);
        }
        buf = malloc (blocksize);
        if (buf == NULL)
            printf ("Erro na alocacao do buffer de transmissao↘
            →");
        memset ( buf, 'T', blocksize );
        resp = gettimeofday(&tv,NULL); // repeti para ↘
        →melhorar a assertividade
        antes = agorinha = tv.tv_sec + tv.tv_usec / ↘
        →1000000.0;
        for (count = 0; count < isize; count++) {
            if (verbose == 2)
                printf ("%s funcao TX para porta %d, bloco %d\n↘
                →",
                    ident,parm->port,count);
            resp = write (sock_connected, buf, blocksize);
            if (resp != blocksize)
                printf ("Erro na gravacao do dado no socket\n")↘
                →;
            if (func == 't')
                read(sock_connected,buf,1); // handshake

            resp = gettimeofday(&tv,NULL);
            agora = tv.tv_sec + tv.tv_usec / 1000000.0;
            tot = agora - antes;
            bps=((count+1) * blocksize * 8) / tot;
            tot = agora - agorinha;
            bpsparc=(blocksize * 8) / tot;
            agorinha = agora;
            if (verbose == 2)
                printf("%s ESTAT #%d,%f,%f,%f\n",ident,count,↘
                →agora,bps,bpsparc);
            parm->taxap = bpsparc;
            parm->taxat = bps;
        }
    }
    if (func == 'r' || func == 'R') { // e' para este no' ↘
    →receber
        if (verbose >= 1) {
            printf ("%s funcao RX para porta %d, %dkbytes\n",↘
            →ident,parm->port,isize);
            if (func == 'R')
```

```
            printf ("%s comunicando sem handshake, ↘
            →blocksize=%d\n",
                    ident,blocksize);
         else
            printf ("%s comunicando com handshake, ↘
            →blocksize=%d\n",
                    ident,blocksize);
      }
      buf = malloc (blocksize);
      if (buf == NULL)
         printf ("Erro na alocacao do buffer de recepcao");
      resp = gettimeofday(&tv,NULL); // repeti para ↘
      →melhorar a assertividade
      antes = agorinha = tv.tv_sec + tv.tv_usec / ↘
      →1000000.0;
      for (count = 0; count < isize; count++) {
         if (verbose == 2)
            printf ("%s funcao RX para porta %d, bloco %d\n↘
            →",
                    ident,parm->port,count);
         resp = read (sock_connected, buf, blocksize);
         if (resp != blocksize)
            printf ("Erro na leitura do dado no socket\n");
         if (func == 'r')
            write (sock_connected,"X",1); // handshake

         resp = gettimeofday(&tv,NULL);
         agora = tv.tv_sec + tv.tv_usec / 1000000.0;
         tot = agora - antes;
         bps=((count+1) * blocksize * 8) / tot;
         tot = agora - agorinha;
         bpsparc=(blocksize * 8) / tot;
         agorinha = agora;
         if (verbose == 2)
            printf("%s ESTAT #%d,%f,%f,%f\n",ident,count,↘
            →agora,bps,bpsparc);
         parm->taxap = bpsparc;
         parm->taxat = bps;
      }
   }

   close (sock_connected);

   resp = gettimeofday(&tv,NULL);
   agora = tv.tv_sec + tv.tv_usec / 1000000.0;

   tot = agora - antes;
```

```
        if (tot == 0)
           bps=0;
        else
           bps=(isize * blocksize * 8) / tot;

        printf ("%s terminou, %10.5f segundos, %6.5fbps\n",ident↘
        →,tot,bps);

        parm->taxap = 0;
        parm->taxat = 0;

        fflush (NULL);

     }
     pthread_exit((void *)0);

  }

  void sig_handler(int sinal) {
     struct timeval tv;
     int count;


     signal (SIGALRM, sig_handler);
     alarm (5);  // seta alarme para cada cinco segundos

     gettimeofday(&tv,NULL);

     if (verbose > 1) {
        printf ("%d",tv.tv_sec);
        for (count = 0; count <= qthreads; count++)
           printf (",%d,%4.2f",parm[count].port,parm[count].↘
           →taxap);
        printf ("\n");
     }

     if (gravar == 1) {
        fprintf (spool,"%d",tv.tv_sec);
        for (count = 0; count <= qthreads; count++)
           //fprintf (spool,",%4.2f,%4.2f",parm[count].taxap,↘
           →parm[count].taxat);
           fprintf (spool,",%9.2f",parm[count].taxap);
        fprintf (spool,"\n");
        fflush(NULL);
     }

  }
```