Implementação de Serviços Diferenciados em uma Rede Local

César Augusto de Oliveira Soares[1]
Rosivelt Alves do Carmo[1]

Orientadores:
Prof. DSc. Joaquim Celestino Júnior[2]
Profa. MSc. Ana Luiza Bessa de Paula Barros Diniz[2]

UECE – Universidade Estadual do Ceará
DEC – Departamento de Estatística e Computação
CEP 60740-000 Fortaleza (CE)
[1]{caos, rosivelt}@larces.uece.br
[2] {celestino, analuiza}@larces.uece.br

LARCES – Laboratório de Redes de Comunicação e Engenharia de Software

Resumo: Para implantação de QoS em uma rede de comutação de pacotes IP tem sido amplamente aceita a arquitetura Diffserv [NWFUSION] [Aroca, 2001], que possui boa escalabilidade, pois divide tarefas entre roteadores de borda e de núcleo, e possui PHBs padronizados e abertos à expansão. Esse artigo visa a análise de um ambiente DiffServ em uma rede local, criado através da ferramenta IPROUTE2, juntamente com a habilitação das opções de QoS no kernel do Linux. Utilizando essa rede, foram feitos testes com o objetivo de verificar o desempenho das ferramentas e do comportamento das disciplinas de serviço, incluindo a verificação das funções de classificação e policiamento dos roteadores de borda e de núcleo de um domínio DiffServ.

Palavras Chaves: QoS, DiffServ, iproute2.

1 Introdução

Mais do que interligação de diferentes redes, cada uma com sua arquitetura própria, pensar em Internet atualmente significa visualização de páginas WEB (sejam elas estáticas ou com algum recurso mais dinâmico), e-mail, transações comerciais, videoconferência, ensino à distância, entre outros.

Por trás de tudo isso, está a transmissão de diversos tipos de mídia, cada qual com os seus respectivos requisitos para se obter um resultado (apresentação para o usuário final) com qualidade: baixo retardo e baixo *jitter* (variação desse atraso) para voz e vídeo (mídias contínuas), e baixas taxas de perda para texto.

Contudo, essa diversidade de tráfego não existia na concepção da Internet, o que fez com que o protocolo de uso predominante na Internet (arquitetura TCP/IP) não se preocupasse em dar um tratamento diferenciado para os fluxos de tráfego, tratando todos da mesma maneira[1], transmitindo os dados sem qualquer garantia de sucesso na sua entrega. Tal comportamento apresentado pelo IP é denominado Serviço de Melhor Esforço (*Best-Effort Service*).

Este serviço é aceitável em situações de baixa carga na rede. Todavia, em momentos de congestionamento pode tornar-se inaceitável a degradação desse serviço (inserção de atraso, perdas e *jitter*) oferecido aos tráfegos.

Com esta mudança de slogan de "IP sobre tudo" para "Tudo sobre IP" [MARTINS], surgiram algumas propostas de implementação de Qualidade de Serviço (QoS) na Internet. Termo este que surgiu com as Telecomunicações e é definido pela ISO como "o efeito coletivo de desempenho que determina o grau de satisfação do usuário de um serviço específico" [Kamienski,1999].

Ao se implementar QoS, muitos conceitos estão envolvidos, como: Que algoritmos de escalonamento e congestionamento aplicar, qual arquitetura para o provimento de QoS será usada, qual a forma de roteamento, etc. Esse conjunto irá definir qual a melhor forma de se associar uma transmissão de qualidade a um preço acessível ao usuário.

Existem várias soluções para fornecer QoS na Internet (MPLS, IntServ, DiffServ etc) [Ferguson, 1998]. No entanto, nesse trabalho, será enfocado apenas o modelo DiffServ (*Differentiated Service*) ou Serviços Diferenciados [RFC2475, 1998], que se caracteriza pela existência de classes de serviços nos roteadores. Assim, as aplicações multimídia, de acordo com as suas características, são encaminhadas para uma dessas classes e, conseqüentemente, têm sua informação enviada com êxito. O método de funcionamento e implementação do DiffServ será abordado com detalhes mais à frente.

Nesse trabalho foi implementada uma rede Diffserv, em uma rede local, utilizando a ferramenta IPROUTE2 [IPROUTE2]. O objetivo é analisar o comportamento dessa rede e a real aplicação das classes de serviço de uma rede Diffserv.

Esse artigo está organizado da seguinte forma: Na seção 1, é feita uma introdução ao conceito de QoS e a sua necessidade, hoje, no mundo da Internet. Na seção 2, um dos modelos de QoS é abordado, no caso, o Diffserv. É dado um esclarecimento sobre o seu funcionamento e sua vantagem em uma rede local. Na seção 3, é apresentada a ferramenta de geração e análise de tráfego bem como o IPROUTE2. São também citados os algoritmos de controle de tráfego utilizados nesse trabalho. Ainda nessa seção, é abordado o porquê de se utilizar o Linux como ambiente de testes e quais ajustes são necessários para se obter uma melhor análise dos resultados. Na seção 4, são apresentados os layouts físico e lógico utilizados, e os testes em si. Para finalizar, na seção 5 são abordadas as últimas considerações do presente trabalho.

2 DiffServ

DiffServ é um modelo de implementação de QoS em redes de computadores, criado pelo IETF, que trabalha com o conceito de classes de serviço e visa agrupar fluxos de tráfego semelhantes em uma mesma classe. Esse modelo possui boa escalabilidade, tornando possível o tratamento de uma grande quantidade de fluxos, por possuir uma baixa granularidade, em provedores backbone. Dessa forma, supre uma das principais deficiências do modelo IntServ (*Integrated Services*) [RFC1633, 1994] que, ao tratar fluxos individuais, torna-se inviável em um backbone, devido a alta carga de sinalização.

---

[1] Na verdade, o protocolo IP possui um campo denominado ToS (*Type of Service*) que possibilita a escolha de alguns serviços como minimizar custos, maximizar vazão e outros.

No Diffserv o tratamento passa a ser baseado em agrupamentos de fluxos, e não em fluxos individuais, o que gera menor carga de sinalização, pois os roteadores tratam grupos de fluxos em vez de fluxos individuais. No entanto, perde-se em eficiência na alocação de recursos, pois o fluxo mais exigente é quem determinará os recursos a serem reservados para toda a classe.

Em uma solicitação de Serviço Diferenciado, um contrato deve ser estabelecido entre usuário e provedor. Nesse contrato, chamado de SLA (*Service Level Agreement*), o usuário compromete-se a gerar um fluxo com as características dispostas no contrato e o provedor compromete-se a encaminhá-lo com a QoS solicitada pelo usuário. Os SLAs são na sua grande maioria estáticos, ou seja, a configuração é feita manualmente, o que não acarreta em sérios problemas na sua administração, visto que, aqui, os contratos possuem um tempo de vida maior do que aqueles feitos no Intserv, onde para cada novo fluxo é criado um SLA .

2.1 PHB

O que caracteriza a forma de encaminhamento dos pacotes é o chamado PHB (*Per Hop Behavior*). O PHB descreve como será realizado o encaminhamento dos pacotes pertencentes a uma mesma classe de serviço. Ele, diferentemente do IntServ, não padroniza serviços, apenas especifica comportamentos de encaminhamento. Essa não padronização permite que os usuários possam escolher entre provedoras que lhe forneçam melhores vantagens. A combinação de PHBs com os parâmetros de caracterização do SLA é que define o serviço a ser fornecido pelo provedor Diffserv. Existem, atualmente, dois PHBs padronizados, o EF (*Expedited Forwarding*) ou encaminhamento expresso [RFC2598, 1999] e o AF (*Assured Forwarding*) ou encaminhamento assegurado [RFC2597, 1999].

O EF é caracterizado por baixos retardo e variação do mesmo, taxa de erros controlada e largura de banda assegurada. É ideal para aplicações que necessitam de rapidez, constância na sua transmissão, com pouco ou nenhum erro, como por exemplo, Telemedicina e telefonia na Internet.

O AF possui, na realidade, níveis distintos de tráfego com níveis variáveis de probabilidade de perda. São quatro as classes AF definidas, com até três níveis de precedência de perda em cada uma. Nesse PHB, não há uma garantia estrita na entrega dos pacotes, o que há é uma garantia de que pacotes marcados como conformes são entregues com alta probabilidade, enquanto que os que excederem a taxa especificada no contrato recebem uma probabilidade de descarte maior, podendo assim, serem mais facilmente descartados em momentos de congestionamento.

2.2 Controle de tráfego

Em uma tecnologia que implementa QoS, é necessário um amplo controle sobre os fluxos que circulam na rede. É preciso que, usuários que solicitam uma determinada garantia nos seus dados, possam ser atendidos sem um mínimo de comprometimento. Para isso, são necessários algoritmos que tanto controlem os congestionamentos como também diferenciem os tráfegos existentes.

Um algoritmo muito conhecido e amplamente usado em implementações Diffserv é o CBQ (*Class-Based Queuing*).

O CBQ é caracterizado por criar classes de uma forma hierárquica onde um certo percentual de banda é fornecido a cada uma. Ele dispõe de dois escalonadores com o objetivo de garantir aos fluxos de tempo real baixos valores de atraso nas filas de espera e ao mesmo tempo a não monopolização por parte deste da ligação. O CBQ visa, ainda, para um maior aproveitamento da ligação, a possibilidade de classes utilizarem um percentual de largura de banda maior do que o fornecido a elas desde que não prejudiquem o fluxo de outras classes. Assim, ele trabalha no sentido de dar prioridade às aplicações mais exigentes, ao fornecer maior percentual de banda, evitando que aplicações menos favorecidas entrem em starvation.

Para entender melhor a estrutura hierárquica do CBQ, um exemplo é mostrado na figura 2.1.

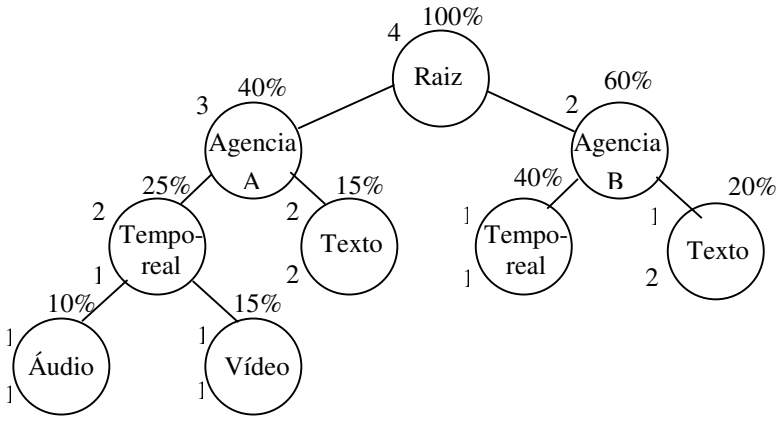

Figura 2.1: Exemplo de estrutura hierárquica do CBQ

No exemplo acima, vemos uma estrutura hierárquica na qual as agências recebem um certo percentual da banda e o distribui entre as suas folhas que são aplicações com requisitos distintos. No canto superior esquerdo, é indicado o seu nível, no canto inferior esquerdo a sua prioridade e os percentuais indicam a quantidade de largura de banda fornecida a cada classe.

Outro modelo de controle de tráfego existente é o TBF (*Token Bucket Filter*). Ele implementa um duplo *token bucket* para o controle da vazão média e da vazão de pico. Seu principal objetivo é atrasar os pacotes que excedam uma determinada especificação e não simplesmente descartá-los. Dentre os diversos parâmetros que podem ser ajustados estão: O *rate* que especifica a vazão média, o *burst* que diz a profundidade do *token bucket* principal, o *peakrate* que ativa o outro *token bucket* (o de pico) e assim, especifica um valor médio, dentre outros.

Uma terceira forma para controle de tráfego é a utilização do algoritmo RED (*Random Early Detection*) que se aproveita dos mecanismos de controle de congestionamento do protocolo de transporte TCP, o qual diminui a taxa de transmissão em vista de perdas de pacotes. O algoritmo RED tem como função a redução do tamanho médio das filas de espera nos roteadores. Esse comprimento é calculado freqüentemente. Se este exceder um valor mínimo determinado, é dada uma certa probabilidade de descarte aos pacotes que chegam ao nó. À medida que aumentar este excedente, a probabilidade de perda de pacotes também aumentará. Sempre que o comprimento médio ultrapassar um valor máximo todos os pacotes serão descartados. Esse mecanismo promove uma distribuição justa das perdas (tráfegos mais agressivos serão mais prejudicados) e procura evitar o efeito de sincronização global.

No entanto, equidade na distribuição dessas perdas não é o propósito dos Serviços Diferenciados. Pelo contrário, algumas classes de tráfego devem ser penalizadas em função de outras. Por isso, um outro algoritmo pertencente ainda à família RED é usado na implementação do PHB AF, denominado GRED (*Generalized RED*). Sua principal diferença em relação ao RED é a existência de várias filas virtuais, cada uma associada ao mesmo conjunto de parâmetros que a fila RED, com a adição de outro parâmetro: prioridade relativa das filas. Dessa forma, diferentes níveis de precedência de descarte podem ser configurados.

3 DiffServ em uma rede local

O modelo DiffServ pode ser implementado em vários tipos de rede. Nesse trabalho, no entanto, é considerada apenas uma rede local.

Alguns caminhos podem ser seguidos para a implantação de uma rede DiffServ em uma LAN, como:

1. Adoção de equipamentos, como roteadores, que estejam aptos e configurados para trabalhar com tal tecnologia
2. Instalação de programas que forneçam à máquina tal capacidade.

Tendo por base que tanto a tecnologia DiffServ se adequa às necessidades da rede na qual será implantada, quanto o custo de administração será viável, no primeiro caminho um considerável custo ainda teria que ser arcado na aquisição e aprendizagem (por parte dos administradores) da configuração dos equipamentos . Na segunda alternativa, pacotes como o iproute2 podem ser obtidos gratuitamente na Internet para a configuração das máquinas que servirão como roteadores, além do fato do suporte à tecnologia DiffServ já fazer parte do kernel do Linux a partir

da versão 2.4.5. Neste caminho, assim como no primeiro, também existe o custo de aprendizagem da ferramenta, mas é adequado o bastante principalmente para pequenas redes, já que a necessidade de equipamentos especializados é reduzida.

A tabela 3.1 contém os programas, suas respectivas funções e versões utilizadas, assim como a versão do kernel do linuix e uma breve descrição das máquinas. Quanto ao processo de instalação, basta seguir os passos do descritos nos respectivos arquivos README que acompanham cada uma dessas ferramentas, não apresentando quaisquer dificuldades.

FERRAMENTAS		
Programa	**Versão**	**Função**
RUDE	Rude-0.60.tar.gz	Geração e recepção de tráfego UDP.
IPROUTE2	Iproute2-21.4.7-now-ss010824.tar.gz	Configuração de rotas, endereços, e classes de serviços.
SISTEMA OPERACIONAL		
Kernel do Linux		**Distribuição**
2.4.18		Slackware 8.1
COMPUTADOR		
- Processador AMD Athlon de 1.2 GHz	- 128 MB de memória	- NIC PCI de 10/100 Mb

Tabela 3.1: Programas, sistema operacional e máquinas utilizadas no testes

3.1 Ferramentas

3.1.1 RUDE E CRUDE

RUDE e CRUDE são dois programas pertencentes a um mesmo pacote. O RUDE é responsável pela recepção de tráfego — inclusive elaboração de estatísticas, como a quantidade de pacotes recebidos e perdidos, jitter máximo e médio, e taxa de transmissão —, e o CRUDE pela geração de tráfego de acordo com um arquivo de script. Nesse arquivo são passados os parâmetros para o tempo de duração e início de cada fluxo, portas de origem e destino, endereço destino, tamanho e taxa de geração do pacote, além da definição do campo TOS do protocolo IPv4.

É possível definir a prioridade de execução destes programas nas máquinas origem e destino, influenciando no escalonamento de processos. Com isso, é possível obter reduções no jitter, uma vez que o processo não terá que esperar muito até que volte a tomar posse da CPU, e assim possa logo tratar o pacote que chegou. Tal fato nos faz relembrar o conceito de "QoS fim a fim", reforçando que todo e qualquer componente participante em uma comunicação influencia, positivamente ou não, na qualidade de serviço final.

Todos os resultados obtidos nos testes foram extraídos a partir das análises feitas pelo CRUDE.

3.1.2 IPROUTE2

Principal elemento para a implantação de uma rede DiffServ, segundo a abordagem escolhida. Até aqui, foram expostos apenas os elementos que geram tráfego sobre uma rede já constituída. Para isso, é preciso a configuração:

1. dos endereços dos hosts, tabelas de roteamento nas máquinas que servirão como roteadores
2. das classes de serviço, policiamento e filtros.

O pacote IPROUTE2 vem com dois comandos principais: ***ip***, utilizado para o item 1 acima; e comando ***tc*** (item 2).

Como a configuração do item 2 exige uma série de passos (definição e remoção de classes, filtros e policiamento), normalmente são criados scripts para facilitar a configuração de ambientes específicos (roteador de borda, núcleo) e manutenção/atualização dos mesmos.

Um tutorial sobre a sua utilização pode ser encontrado em [IPROUTE2].

3.2 Ambiente necessário

Todos os programas até aqui descritos fazem uso do sistema operacional Linux que, intrinsecamente, a partir do kernel 2.4.5, provê suporte à qualidade de serviço, disponibilizando diversas políticas de escalonamento de filas (CBQ, TBF, RED, GRED...), suporte ao RSVP (protocolo de sinalização de reserva de recursos, utilizado em redes integradas - IntServ), além da marcação do campo DSCP (responsável por indicar o PHB pelo qual o pacote deve ser tratado) e suporte à arquitetura DiffServ.

Essas opções devem estar selecionadas antes da compilação do kernel e, além disso, duas outras mudanças no núcleo são necessárias para melhorar a granularidade temporal do Linux. É sabido que a constante interna HZ que representa a freqüência do temporizador interno do Linux tem como valor padrão 100 em arquiteturas x86. Isso implica num limite inferior de intervalo de tempo de 10ms para o escalonamento de eventos e processos, como também afeta a precisão de medição de tempo usada no controle de QoS e controle de tráfego [Prior, 2001].

Como principal conseqüência de tal limitação está o possível surgimento de um elevado *jitter*, já que tanto o transmissor poderá ser interrompido em intervalos múltiplos de 10ms, como o receptor ficará impossibilitado pelo mesmo tipo de intervalo de atender o tráfego que está sendo recebido.

Por outro lado, poderia se dizer que um alto overhead seria gerado pela mudança de contexto (ato de salvar os registradores e variáveis que indicam o estado de um processo em execução, como o *contador de instrução*, *registrador de instrução* e *endereços*, pilha etc., e substituí-lo por um outro), se essa constante tivesse seu valor aumentado para 1024, reduzindo assim para 1ms o intervalo de tempo para o escalonamento de processos. A partir do processador Pentium, tal mudança já pode ser suportada, contudo.

Essa constante encontra-se no caminho *include/asm/param.h* a partir do diretório do código fonte do Linux.

A segunda mudança está na *forma* de medição, e não mais no menor intervalo de tempo. A constante PSCHED_CLOCK_SOURCE, localizada em *include/net/pkt_sched.h* a partir do diretório do código fonte do Linux, indica a forma de medição do tempo, tendo como valor padrão PSCHED_JIFFIES o qual, segundo [Prior, 2001], pode tornar o policiamento não confiável. A maneira mais eficiente de medição de tempo é alterar valor para PSCHED_CPU, que se baseia em contadores internos do microprocessador, estando restrita a algumas arquiteturas, já que estas devem implementar o TSC (*TimeStamp Counter*). Praticamente todos os processadores da família x86 a partir do Pentium e da família Alpha o implementam [Prior, 2001].

As opções de compilação a serem selecionadas encontram-se em [Freitas, 2002].

3.3 Controle de tráfego no Linux

O controle de tráfego no Linux baseia-se em torno de 4 conceitos-chave: disciplina de serviço, classe, filtro e policiamento. Este controle é bastante flexível, permitindo uma série de combinações, devido a sua arquitetura de caráter recursivo (figura 3.1). Tais conceitos são descritos a seguir.

Disciplina de serviço é a forma de gerenciamento da(s) fila(s) de tráfego, podendo ela ser dotada de um algoritmo de escalonamento de filas, o que acontece na maioria dos casos (como CBQ, TBF, RED...) ou apenas exercer algumas funções de classificação. Nesse caso chamadas de *pseudo-disciplinas*, como *dsmark* cuja função principal é a obtenção e (re)marcação do campo DSCP dos pacotes, campo esse responsável por indicar o PHB pelo qual o pacote será tratado. As disciplinas de serviço mais complexas contêm filtros e classes.

Classe é um nó na hierarquia de controle de tráfego (figura 4.1). Uma classe não é responsável pelo gerenciamento das filas. Para isso ela recorre a uma disciplina de serviço, sendo possível um grande número de combinações a partir dessa recursividade. Assim, é possível a criação de uma hierarquia onde em cada nó folha esteja instalada uma disciplina de serviço diferente. Esses nós são os verdadeiros responsáveis pelo controle do tráfego em si.

Essa flexibilidade é mais notável quando usada a disciplina de serviço CBQ, a qual permite a criação de várias classes, ao contrário de outras disciplinas, como TBF, RED, GRED etc.

Filtro é o mecanismo através do qual um pacote é atribuído a uma classe. Um filtro é chamado no momento que um pacote chega a uma disciplina de serviço. Cada disciplina ou classe contém uma lista de filtros, distintos entre si pelo protocolo ao qual se aplicam (IP, ICMP...) e sua prioridade. Um exemplo de filtro bastante útil nos testes é o *u32*, capaz de efetuar a classificação através de qualquer campo de um pacote, abrangendo os protocolos IPv4, IPv6, UDP, TCP e ICMP na versão do IPROUTE utilizada.

Policiamento é qualquer atitude tomada frente à chegada de um pacote. Diversas ações são possíveis de acordo com o perfil do pacote. Por exemplo, se chega um pacote ao roteador a uma taxa acima da permitida, poderá ser descartado, ter sua prioridade diminuída, sendo atribuído à classe de serviço de melhor esforço, ou passar por um processo chamado moldagem de tráfego, no qual um pacote fica numa fila de espera até que o atraso introduzido faça com que ele volte a ter uma taxa de transmissão aceitável, estando assim dentro do perfil.

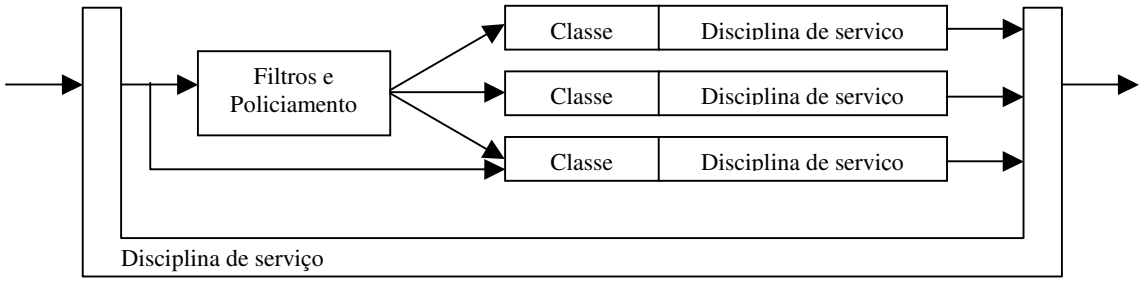

Figura 3.1: Arquitetura do controle de tráfego no Linux

3.3.1 DISCIPLINA DE SERVIÇO CBQ

A disciplina CBQ (*Class Based Queuing*) permite a criação de inúmeras classes de forma hierárquica, sendo esta sua principal característica. Cada uma dessas classes pode fazer recorrência a uma nova disciplina de serviço, conforme a arquitetura de controle de tráfego do Linux (figura 3.1). Cada classe pode ser de três tipos:

- *Shared* (opção padrão) → caso uma classe com tal configuração tenha banda que não esteja sendo utilizada, essa banda será compartilhada com outras classes que estejam "sobrecarregadas" de fluxos. Caso a banda configurada seja menor que a da classe imediatamente superior na hierarquia, o limite "real" de banda será igual ao da classe pai.
- *Bounded* → essa opção faz com que a classe seja sempre limitada à banda que lhe foi atribuída. Assim como em shared, essa opção também faz com que sua banda seja compartilhada, caso essa esteja disponível, para outras classes que estejam precisando.
- *Isolated* → essa opção limita a classe à banda que lhe foi atribuída, como também proíbe o compartilhamento da sua banda com outras classes. Tal opção faz com que a classe também não seja mais limitada por sua classe pai, mesmo que esta seja *bounded*, passando a consumir banda enquanto houver disponível, ou seja, enquanto nenhuma outra classe esteja insatisfeita. O tráfego atribuído a essa classe também não é debitado da sua classe pai, ao contrário das outras opções. No entanto, pelo menos na versão utilizada, essa opção acaba introduzindo alguma instabilidade nos resultados.

Ao mesmo tempo em que as classes obedecem a uma hierarquia, elas possuem uma certa independência no sentido de que uma classe filha não obrigatoriamente deve possuir uma capacidade menor que sua antecessora, ou que a capacidade da classe pai deve ser igual à soma das capacidades de suas descendentes.

3.3.2 DISCIPLINA DE SERVIÇO TBF

Como o próprio nome sugere (*Token Bucket Filter*), essa disciplina baseia-se no algoritmo de *token bucket* para modelagem e controle das taxas de transmissão média e máxima, sendo portanto necessária a utilização de 2 *token buckets*.

A análise desse algoritmo é feita através de uma analogia com um balde com fichas (*tokens*), onde cada ficha significa permissão para transmitir uma certa quantidade de tráfego, que pode ser medida em bytes ou em pacotes.

Assumindo que para cada ficha corresponde a 1 byte, um pacote só poderá ser transmitido quando houver uma quantidade de fichas igual ao seu tamanho. Caso contrário, o pacote deverá esperar em uma fila (*buffer*) até que as fichas necessárias sejam geradas. A cada transmissão, as fichas correspondentes são eliminadas do balde, mas também estão sendo geradas a uma taxa constante *r*.

Com uma capacidade do balde (*bucket*) igual a *b tokens*, e assumindo que o tráfego está sendo gerado a uma taxa de pico *p,* é possível transmitir em um intervalo de tempo *T*, uma quantidade de dados igual ao menor valor entre {*b+rT*, pT} [Mota, 2000].

A modelagem ocorre quando o tráfego chega ao roteador numa taxa maior que a da geração de *tokens* e também não há *tokens* disponíveis no balde, sendo obrigado a esperar em uma fila (buffer). Como esta também é finita, os pacotes que chegarem quando a fila de espera estiver cheia serão descartados.

Há também períodos de rajadas (*bursts*), principalmente após um período de inatividade na rede, quando *tokens* são produzidos e armazenados no balde, mas sem serem consumidos, estando portanto disponíveis para pacotes que cheguem no roteador a uma taxa de pico *p*.

4 Testes

Antes de testar o controle de tráfego do Linux, sobretudo as disciplinas de serviço CBQ e TBF, foram feitos alguns testes com as ferramentas de geração e recepção de tráfego (RUDE e CRUDE), de forma a conhecer os tipos de influências que poderiam causar sobre os resultados.

Concluída a primeira fase, foram feitos testes com as disciplinas CBQ e TBF, a fim de determinar a diferença entre a taxa máxima de transmissão configurada e a taxa real permitida, avaliando assim sua precisão. Tais configurações foram feitas na máquina *Apolo* (figura 4.2) que, de acordo com o arranjo lógico, servirá como roteador.

Os testes basearam-se em [Prior, 2001], mas para avaliar melhor o comportamento das disciplinas de serviço, uma nova seqüência de testes foi estabelecida , dividida em 3 etapas:

- Etapa 1: Geração de apenas 1 fluxo, dobrando a taxa de policiamento a cada teste, partindo de 100Kbps até 1.6 Mbps. Foi criada uma hierarquia simples de classes, onde a classe pai era sempre *bounded* a 10Mbps (capacidade da rede), e uma classe filha onde era modificada a taxa de policiamento. Nos testes com TBF, esta última passava a conter tal disciplina (figura 4.1).
- Etapa 2: Mesma hierarquia de classes, com a mesma seqüência de taxas de policiamento, mas agora com o aumento do número de fluxos (2 e 4 fluxos).
- Etapa 3: Aumento do número de classes, cada uma com apenas um fluxo, novamente utilizando a seqüência de taxas de policiamento.

Em cada uma dessas etapas, foi gerado tráfego UDP, com um tamanho de pacote de 750 bytes e a uma taxa de 2Mbps por fluxo.

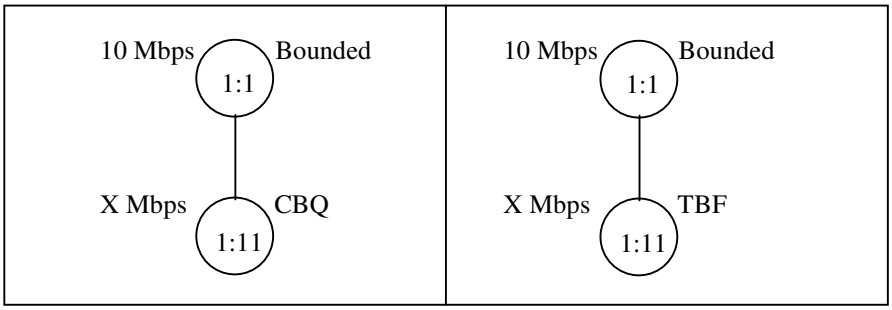

Figura 4.1: Hierarquia de classes para a primeira e segunda etapas dos testes das disciplinas de serviço.

Até então, o *Apolo* ainda não podia ser caracterizado como um roteador, seja de borda (*Edge router*) ou de núcleo (*Core router*), de um domínio DiffServ, pois lhe faltava a definição de ações de policiamento, como o descarte de pacotes ou a atribuição do mesmo a uma classe de prioridade inferior. Também não estavam definidos os PHBs padrões, como *Assured Forwarding* (configurado a partir da disciplina GRED). Com tais adições, tem-se então um roteador de núcleo, já que este considera que todos os pacotes estão com o campo DSCP previamente e corretamente marcado.

Por fim, seria adicionada a esse roteador a capacidade de tratar pacotes cujo campo DSCP não estivesse previamente marcado, ou com uma marcação inválida, tomando, portanto, a postura de um roteador de borda. Nos

testes, tais pacotes são atribuídos à classe BE (*Best Effort* – Melhor Esforço). Seguindo a classificação MF (*Multi-field*), que pode se basear em vários campos do protocolo, pacotes originários da porta 3025, com o endereço de origem 192.126.193.2 são atribuídos à classe EF (*Expedited Forwarding*), enquanto pacotes com destino a 192.168.225.2 são atribuídos à classe AF2.

4.1 Configuração física

A configuração física utilizada nos testes é constituída de três computadores pessoais, todos ligados a um switch *ethernet* a 10Mbps. A presença de um switch assegura-nos que as mensagens não se propagam para outras máquinas (caso fosse utilizado um *hub*) o que geraria uma carga na rede maior que a esperada.

Em todos os hosts deve ser instalado o pacote *iproute2*, e devem ser feitas as devidas modificações no kernel. Com o iproute2, são feitas as configurações de endereço e rotas (isso também pode ser feito através de outros comandos próprios do Linux, como o *route*).

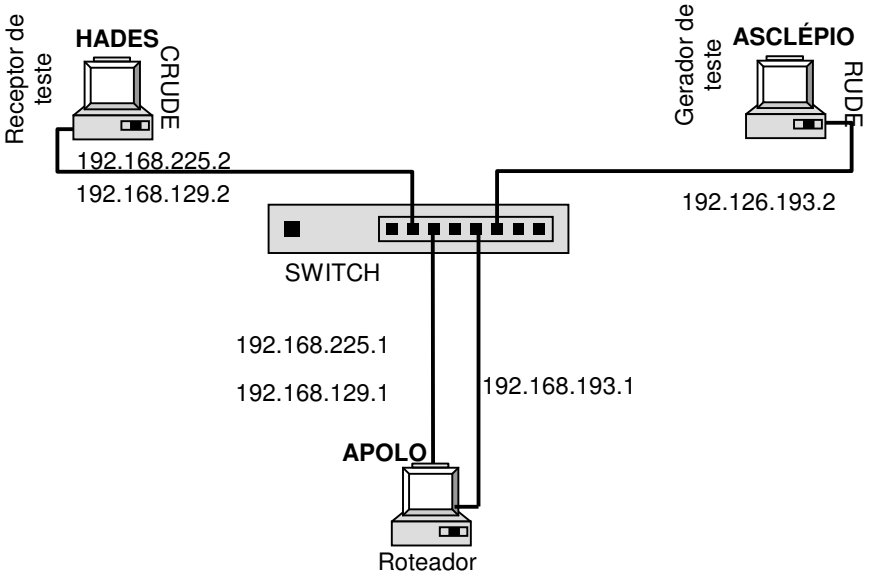

Figura 4.2: Configuração física

4.2 Configuração lógica

Inicialmente, foram criadas duas redes distintas (192.126.193.0/24 e 192.168.129.0/24) com o roteador conectando as duas. Posteriormente, os testes apresentaram a necessidade de criação de uma terceira (192.168.225.0/24) (figura 4.3). O motivo para tal acréscimo encontra-se explicado no tópico sobre os testes do roteador de borda.

Todo tráfego gerado nos testes teve sempre como origem e destino redes diferentes, de forma a passar pelo *Apolo*, e assim este fazer o devido controle de tráfego, já que neste encontram-se definidas as classes de serviço.

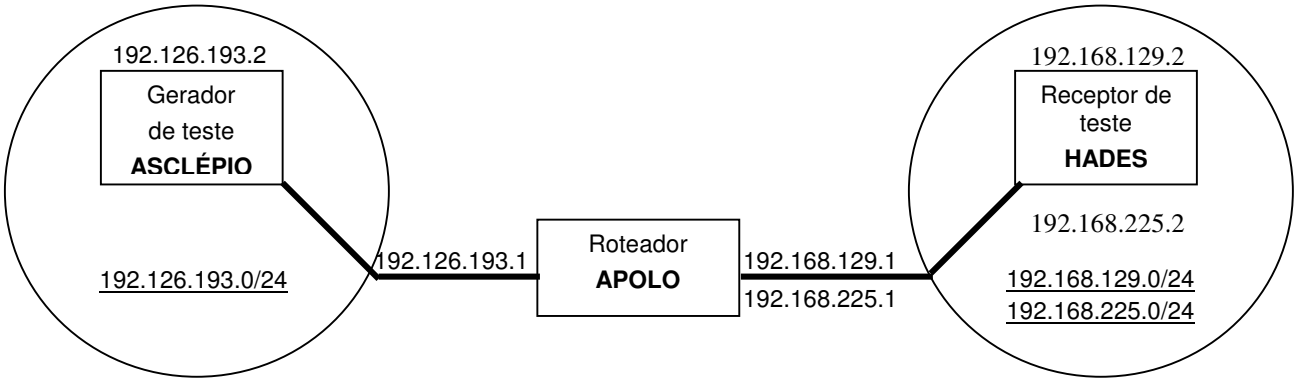

Figura 4.3: Configuração lógica

4.3 Testes à ferramenta geradora e receptora de tráfego

O objetivo da primeira fase é testar o quão preciso o RUDE é em termos de geração de fluxos, procurando saber suas limitações e sensibilidades a mudanças nas características dos tráfegos a serem gerados.

Nos testes, foram analisadas as alterações de 4 variáveis:

- Taxa de transmissão
- Tamanho do pacote
- Quantidade de fluxos gerados
- Prioridade de execução do processo

Uma última variável foi posteriormente acrescentada, devido a alguns resultados incoerentes obtidos ao aumentar a quantidade de fluxos gerados: a ordem de declaração no arquivo de script dos fluxos a serem gerados. Foi notado que, pelo menos na versão utilizada do RUDE, os primeiros fluxos a serem declarados obtinham uma certa prioridade em relação aos demais durante envio pelo transmissor, consumindo uma maior largura de banda.

Para efeito de exemplificação dos resultados obtidos, são mostradas a seguir as principais tabelas baseadas na saída do programa CRUDE, tendo como transmissor *Asclépio*, gerando 500 pacotes/seg de 64 bytes cada, e receptor *Apolo*. *Flow_ID* é apenas uma identificação que o RUDE usa para distinguir os fluxos.

Uma outra observação: quando mencionado "escalonamento prioritário", isso diz respeito à prioridade de execução dos processos RUDE e CRUDE tanto nos hosts origem como destino.

Flow_ID = 30					
Escalonamento	**Vazão**	**Qtd. Pacotes**	***Jitter* médio**	***Jitter* máximo**	**Perdas**
Normal	32 Kbps	30001	0,004 ms	0,918 ms	0
Prioritário	32 Kbps	30001	0,004 ms	0,495 ms	0

Tabela 4.1: Geração de 1 fluxo apenas, sem carga adicional na rede

	ESCALONAMENTO			
	NORMAL		**PRIORITÁRIO**	
	FLOW ID = 25	FLOW ID = 30	FLOW ID = 25	FLOW ID = 30
Vazão	32 kbps	32 kbps	32 kbps	32 kbps
Qtd. Pacotes	29996	29996	30000	30001
***Jitter* médio**	**0,012 ms**	**0,016 ms**	**0,004 ms**	**0,009 ms**
***Jitter* máximo**	25,28 ms	25,37 ms	0,893 ms	1,01 ms
Perdas	2	0	0	0

Tabela 4.2: Geração de 2 fluxos com características iguais, sem carga adicional na rede

	ESCALONAMENTO			
	NORMAL		**PRIORITÁRIO**	
	FLOW ID = 25	FLOW ID = 30	FLOW ID = 25	FLOW ID = 30
Vazão	36,49 kbps	32 kbps	36,49 kbps	32 kbps
Qtd. Pacotes	34208	30001	34208	30001
***Jitter* médio**	0,012 ms	0,010 ms	0,012 ms	0,010 ms
***Jitter* máximo**	**0,490 ms**	**0,561 ms**	**0,936 ms**	**1,06 ms**
Perdas	0	0	0	0

Tabela 4.3: Geração de 2 fluxos diferentes, sem carga adicional na rede

A principal conclusão dessa fase é que ***todos*** os elementos intermediários em uma comunicação entre duas máquinas afetam a qualidade de serviço oferecida ao usuário final. O fator mais relevante foi o escalonamento dos processos nas máquinas origem e destino. Ele afeta diretamente o *jitter*.

Isso se deve ao fato de que, quando o processo gerador não está sendo executado de forma prioritária (a ferramenta RUDE oferece essa opção, sendo o intervalo 1-90 usado para tal configuração, sendo o valor 1 o de maior prioridade), apesar dele ter gerado pacotes a serem transmitidos, ele terá que aguardar até que "tome posse" da cpu para poder fazê-lo. Esse intervalo pode variar bastante, afetando assim o *jitter*, de acordo com o número de processos, incluindo os *daemons* do sistema, que estão sendo executados na máquina.

Na recepção do tráfego, o fato ocorre de forma análoga: embora os pacotes já tenham chegado, o processo de recepção (a ferramenta CRUDE, no nosso exemplo) não pode tratá-los pelo mesmo motivo, ou seja, ainda não foi escalonado.

4.4 Testes à disciplina de serviço CBQ

Seguindo as fases de teste estabelecidas para as disciplinas de serviço, nessa primeira foi criada uma hierarquia de acordo com a figura 4.1, onde na classe filha as taxas de policiamento foram variando de acordo com a tabela 4.4, lembrando ainda que o fluxo foi gerado a 2Mbps.

Classe e taxa de policiamento			**Fluxo gerado**	**Vazão**	**Jitter médio**	**Perda (pacotes)**
1:1B[2] 10 Mbps	1:11B	100Kbps	2 Mbps	106 Kbps	6,66 ms	18864
		200Kbps		223 Kbps	2,65 ms	17703
		400Kbps		498 Kbps	2,31 ms	14956
		800Kbps		1249 Kbps	1,58 ms	7058
		1.6 Mbps		2048 Kbps	0,009 ms	0

Tabela 4.4: Quadro resumo dos testes da 1ª fase do CBQ

De acordo com os resultados, essa disciplina mostrou-se incapaz de efetuar um policiamento com precisão sobre o tráfego selecionado. À medida que a taxa configurada foi duplicada, a tolerância também aumentou, não na mesma proporção, mas ainda assim em um valor inaceitável.

[2] O modelo superior:inferior diz respeito a forma de distinção entre disciplinas de serviço e classes no Linux. A parte superior refere-se à disciplina e a inferior à classe pertencente a essa disciplina. O símbolo "B" especifica que a classe é Bounded. (Ver seção 3.3)

Classe e taxa de policiamento			Fluxos gerados	Número de fluxos	Vazão total
1:1B 10 Mbps	1:11B	100 Kbps	2 Mbps	2	102 Kbps
				4	122 Kbps
		200 Kbps		2	302 Kbps
				4	278 Kbps
		400 Kbps		2	523 Kbps
				4	497 Kbps
		800 Kbps		2	1,176 Mbps
				4	1,024 Mbps
		1,6 Mbps		2	2,362 Mbps
				4	2,173 Mbps

Tabela 4.5: Quadro resumo dos testes da 2ª fase do CBQ

Quanto à segunda fase, o aumento do número de fluxos inseriu uma certa instabilidade na variação de tolerância dessa disciplina, que tornou-se mais precisa quando 2 fluxos foram gerados a uma taxa de 100Kbps, e o contrário acontecendo à medida que a taxa de transmissão foi aumentando, sendo mais precisa para o tratamento de 4 fluxos. De qualquer forma, tais resultados em nada encorajam a utilização dessa disciplina (*tabela 4.5*).

Classe e taxa de policiamento			Fluxo gerado por classe	Número de classes	Vazão média por classe
1:1B 10 Mbps	1:11B	100 Kbps	2 Mbps	2	102 Kbps
				4	103 Kbps
		200 Kbps		2	206 Kbps
				4	209,36 Kbps
		400 Kbps		2	420 Kbps
				4	445,31 Kbps
		800 Kbps		2	887 Kbps
				4	930,7 Kbps
		1,6 Mbps		2	1,896 Mbps
				4	2 Mbps

Tabela 4.6: Quadro resumo dos testes da 3ª fase do CBQ

Por fim, o aumento do número de classes previsto pela terceira fase também contribuiu negativamente na precisão de policiamento do CBQ, mostrando-se menos tolerante quanto menor o número de classes (tabela 4.6).

Como conclusão dos testes sobre a disciplina de serviço CBQ, ficou claro que esta não pode ser indicada, pelo menos na versão do IPROUTE2 utilizada, para policiamento de tráfego, tendo como melhor aplicação a criação de hierarquias de classes, com os nós folha sendo servidos por uma outra disciplina que não a CBQ.

4.5 Testes à disciplina de serviço TBF

Bem mais entusiasmante que CBQ, essa disciplina mostrou-se bem mais confiável e determinística no policiamento de tráfego, mantendo esse comportamento nas 3 fases de testes definidas para as disciplinas de serviço.

Classe e taxa de policiamento			Fluxo gerado	Vazão	Jitter médio	Perda
1:1B 10 Mbps	1:11B	100Kbps	2 Mbps	97 Kbps	1,81 ms	19055
		200Kbps		194,46 Kbps	1,38 ms	18082
		400Kbps		388,566 Kbps	0,983 ms	16144
		800Kbps		776,791 Kbps	1,38 ms	12263

Tabela 4.7: Quadro resumo dos testes da 1ª fase do TBF

O fluxo, além de não ultrapassar a taxa de transmissão permitida, manteve-se policiado em, aproximadamente, 97% desse valor (tabela 4.7).

Classe e taxa de policiamento			Fluxo gerado	Número de fluxos	Vazão total
1:1B 10 Mbps	1:11B	100 Kbps	2 Mbps	2	97,45 Kbps
				4	97,69 Kbps
		200 Kbps		2	194,79 Kbps
				4	194,75 Kbps
		400 Kbps		2	388,68 Kbps
				4	388,7 Kbps
		800 Kbps		2	776,86 Kbps
				4	777 Kbps

Tabela 4.8: Quadro resumo dos testes da 2ª fase do TBF

O aumento do número de fluxos (tabela 4.8) também não interferiu no grau de precisão do TBF, gerando apenas uma ligeira variação nos resultados, que ora aumentava com o número de fluxos, ora diminuía, mostrando-se, portanto, dentro de limites toleráveis de policiamento.

Classe e taxa de policiamento			Fluxo gerado por classe	Número de classes	Vazão médio por classe
1:1B 10 Mbps	1:11B	100 Kbps	2 Mbps	2	97,41 Kbps
				4	97,41 Kbps
		200 Kbps		2	194,45 Kbps
				4	194,45 Kbps
		400 Kbps		2	388,55 Kbps
				4	388,56 Kbps
		800 Kbps		2	776,78 Kbps
				4	776,78 Kbps

Tabela 4.9: Quadro resumo dos testes da 3ª fase do TBF

Novamente, na terceira fase, o TBF continuou com sua mesma precisão e, se comparados os resultados entre todas as fases, pode-se verificar que a variação dos resultados foi na ordem de décimos (tabela 4.9).

4.6 Roteador de borda

Quanto ao roteador de borda, foi testada a sua atuação na marcação de pacotes cujo DSCP possuía um valor desconhecido ou inválido, como também o tratamento de pacotes previamente marcados, seguindo a definição de valores do valor de DSCP para os respectivos PHBs, como se segue no quadro abaixo:

AF (Assured Forwarding)				
PRECEDÊNCIA DE PERDA	CLASSE 1	CLASSE 2	CLASSE 3	CLASSE 4
Baixa	001010	010010	011010	100010
Média	001100	010100	011100	100100
Alta	001110	010110	011110	100110
EF (Expedited Forwarding)			**BE (Best Effort)**	
101110			000000	

Tabela 4.10: Valores de DSCP para os PHBs padrão

Para tais fluxos foram estabelecidos os seguintes padrões:

- Aqueles que fossem originados pela porta 3025, e endereço de origem 192.126.193.2, foram atribuídos ao serviço EF (*Expedited Forwarding*), com uma taxa de 50Kbps.
- Aqueles destinados ao endereço 192.168.225.2 foram classificados como pertencentes ao serviço AF2 (*Assured Forwarding*), com uma taxa de até 100Kbps.

Com relação aos pacotes previamente marcados, seguem-se as seguintes definições para o valor de DSCP:

- 0xB8, serviço EF, com uma taxa de 245 kbps.
- 0x00, serviço de melhor esforço (*BE – Best Effort*) e taxa de 200Kbps.
- Para valores do serviço AFX, a taxa será de 375 Kbps.

Todos os fluxos tiveram 192.126.193.2 como endereço origem, portanto tal valor não é mostrado na tabela 4.11 com os resultados dos testes. As linhas com um *, ainda na mesma tabela, são as que provocaram uma situação de conflito no momento de classificação do pacote. Tais situações são explicadas adiante.

PACOTES NÃO MARCADOS					
ID DO FLUXO	TOS	END. DESTINO	PORTA DE ORIGEM	CLASSIFICAÇÃO RESULTANTE	VAZÃO[3]
10	—	192.168.129.2	3025	EF	48,5 Kbps
15	—	192.168.225.2	3000	AF2	97,8 Kbps
10*	—	192.168.225.2	3025	EF	48,5 Kbps
PACOTES MARCADOS					
10*	0xB8[4]	192.168.225.2	3000	AF2	97,8 Kbps
10	0xB8	192.168.129.2	3000	EF	232,95 Kbps
10	0x00	192.168.129.2	3000	BE	209,36 Kbps

Tabela 4.11: Quadro resumo dos testes com o roteador de borda

Algumas considerações ainda devem ser feitas acerca do funcionamento do IPROUTE2. De acordo com a configuração lógica e as condições de classificação citadas acima, pode-se notar um certo conflito de classificação quando ambas as condições são satisfeitas, ou seja, um fluxo originário da porta 3025, do host 192.126.193.2, com destino a 192.168.225.2. Neste caso, foi notado que tal configuração resultava na escolha da primeira opção, ou seja, a atribuição do tráfego ao serviço EF.

Foi percebido também um outro tipo de conflito, resultado da geração de um pacote com DSCP previamente marcado (como exemplo, 0xB8, valor para EF) com destino a 192.168.225.2. Como conseqüência, o pacote foi atribuído à classe AF2. O motivo para tal, acredita-se ser o fato de que os filtros para pacotes não marcados estão definidos na disciplina de serviço *INGRESS*, sendo os primeiros a serem verificados. Como a condição de verificação é válida, o pacote é repassado para a classe associada ao filtro, que por sua vez possui uma disciplina de serviço, que fica a cargo da gerência da fila de espera, neste caso AF2.

Para tentar forçar uma classificação ao serviço EF, e que o tráfego continuasse destinado ao mesmo host, um segundo IP (192.168.225.3) foi atribuído a essa máquina (o Linux permite que uma interface possua mais de um endereço IP) e então, mudamos o destino do fluxo para tal endereço. Todavia, a classificação permaneceu a mesma. Outros endereços pertencentes à mesma rede foram testados sem sucesso. Somente quando foi estabelecido um IP de uma terceira rede (192.168.129.2) é que foi possível ao fluxo ser servido por EF. Nota-se portanto, que a versão utilizada do IP, juntamente com os scripts utilizados, não diferencia endereços de uma mesma rede no momento de classificação dos pacotes.

Nos demais casos, onde não havia conflitos, os pacotes foram classificados com êxito, apresentando valores já esperados, uma vez que as disciplinas tinham sido estudadas anteriormente, e tiveram seus comportamentos discutidos e supramencionados.

4.7 Roteador de núcleo

De acordo com a própria arquitetura DiffServ, a funções de um roteador de núcleo resumem-se à atribuição do pacote à classe que lhe proverá o devido serviço e, é claro, o encaminhamento do mesmo, por motivo de escalabilidade, deixando as tarefas que exigem maior processamento para os nós com menor carga (roteador de borda), e ficando restrito a essas simples atividades, uma vez que este tipo de nó é, visto de regra, sobrecarregado.

Logo, um roteador de núcleo assume que os fluxos estão dentro dos padrões estabelecidos nos contratos (*SLAs*), não executando ações de policiamento.

Com essas características, as taxas de transmissão dos pacotes não poderiam ser superiores às configuradas nestes nós. Como os testes de classificação dos pacotes já foram feitos, inclusive com taxas que superavam às permitidas nos nós, disparando, assim, ações de policiamento, os testes para análise do comportamento de um roteador de núcleo mostraram-se desnecessários.

---

[3] Os fluxos novamente foram gerados a 2Mbps.

[4] Valor hexadecimal de 10111000, considerando-se os bits CU (*Currently Unused*) iguais a zero.

5 Considerações finais

Esse artigo teve como objetivo visualizar a possibilidade de implantação de uma rede DiffServ em uma rede local, a partir de uma solução a nível lógico, através da instalação do pacote IPROUTE2 nos nós que funcionaram como roteador, seja de borda ou núcleo, e compilação do kernel do Linux com as opções de QoS habilitadas [Freitas, 2002].

A partir das fases de teste estabelecidas, os resultados mostraram que:

- Apesar da disciplina de serviço CBQ mostrar-se bastante útil e flexível na criação de hierarquia de classes, não é adequada para policiamento de tráfego, devido ao aumento de tolerância apresentado ao crescer o valor da taxa de policiamento.
- A disciplina de serviço TBF tem uma melhor precisão no policiamento do tráfego, permanecendo aproximadamente a 97% da taxa configurada, mesmo quando esse valor é aumentado como demonstrado nos testes. Podendo, então, ser usada para o serviço EF.

Para o *Assured Forwarding*, o gerenciamento das filas fica a cargo do GRED, uma vez que esta trabalha com diferentes níveis de precedência de descarte, característica exigida por esse PHB. Já para o serviço de melhor esforço, uma disciplina útil é a RED.

Portanto, a possibilidade de implantação dessa rede existe, embora ainda seja necessário testar essas disciplinas em situações mais reais, nas quais aplicações multimídia utilizem essa rede como base de transporte de seus dados, sejam com padrões de vídeo, voz ou texto.

6 Referências bibliográficas

[IPROUTE2] "IPROUTE2 Utility Suite Howto". Disponível em http://www.linuxgrill.com/iproute2-toc.html

[NWFUSION] "TCP/IP and QoS". Fonte: Network World Newsletters. http://www.nwfusion.com/. Também disponível em <http://www.anixter.co.uk/.../525835216F46182780256B870050D899/$FILE/TCP_IP and QOS basics.PDF

[MARTINS] Martins, J. "Qualidade de Serviço (QoS) em redes IP – Princípios Básicos, Parâmetros e Mecanismos". Disponível em <http://www.networkdesigners.com.br/Artigos/qos/qos.html>

[Ferguson, 1998] Ferguson, P., Huston, G. "Quality of Service: Delivering QoS on the Internet and in Corporate Networks", 1998. Wiley Computer Publishing.

[RFC1633, 1994]* Braden, R., Clark., D., Shenker, S. "Integrated Services in the Internet Architecture: an Overview", 1994.

[RFC2475, 1998]* Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W. "Architecture for Differentiated Services", 1998.

[RFC2597, 1999]* Heinanen, J., Baker, F., Weiss, W., Wroclawski, J. "Assured Forwarding PHB Group", 1999.

[RFC2598, 1999]* Jacobson, V., Nichols, K., Poduri, K. "An Expedited Forwarding PHB", 1999.

[Kamienski,1999] Kamienski, C. A. "Qualidade de Serviço na Internet". Centro Federal de Educação Tecnológica da Bahia. Disponível em <http://www.cin.ufpe.br/~cak/publications/ kamienski-qos-eine-99.pdf>

[Mota, 2000] Mota, O.T.J.D.D.L. "Internet com Qualidade: Um Estudo das Propostas de Diferenciação de Serviços", 2000. Pontifícia Universidade Católica do Rio de Janeiro.

[Aroca, 2001] Aroca, R. V. "Um Estudo sobre Qualidade de Serviço em Redes IP", 2001. Associação das Escolas Reunidas – ASSER. Disponível em <http://www.geocities.com/rafaelaroca/qos/ip-qos-paper.pdf>

[Prior, 2001] Prior, R. P. M. C. "Qualidade de Serviço em Redes de Comutação de Pacotes", 2001.

[Freitas, 2002] Freitas, A. E. S. "Roteamento avançado com o Linux", 2002. Disponível em < http://www.rnp.br/newsgen/0201/roteamento_linux.shtml>

* As RFCs estão disponíveis no endereço http://www.itac.gr.jp/rfc/rfc.asp