Incorporação de Qualidade de Serviço em Aplicações Telemáticas

Rossano P. Pinto,* Eliane G. Guimarães†
Eleri Cardozo, Mauricio F. Magalhães
DCA-FEEC - UNICAMP
CP 6101 - Campinas-SP 13083-970
{rossano,eliane,eleri,mauricio}@dca.fee.unicamp.br

Resumo

Serviços telemáticos sobre a Internet demandam a garantia de parâmetros de tráfego tais como banda assegurada e baixo jitter. Algumas arquiteturas foram propostas visando incorporar qualidade de serviço às redes TCP/IP. Dentre estas arquiteturas, as arquiteturas de Serviços Integrados (IntServ) e de Serviços Diferenciados (DiffServ) são dignas de nota. A arquitetura DiffServ é mais escalável que a arquitetura IntServ, razão pela qual foi escolhida para a nova geração da Internet (Internet 2). Este artigo descreve um mecanismo para incorporação de qualidade de serviço em aplicações telemáticas fazendo uso de um interceptador de qualidade de serviço (IQoS) e um *Bandwidth Broker* DiffServ para gerência de banda em um domínio DiffServ.
Palavras-chave: Arquitetura DiffServ, *Bandwidth Broker*, QoS Broker, CORBA, Linux QoS.

Abstract

Telematic services over the Internet demand the enforcement of traffic parameters such as assured bandwidth and low jitter. Some architectures were proposed in order to handle quality of service in TCP/IP networks. Among these architectures, the Integrated Services (IntServ) and the Differentiated Services (DiffServ) architectures are worth of mention. The DiffServ architecture is more scalable than the IntServ architecture, the reason why it was chosen for the next generation Internet (Internet 2). This paper describes a mechanism that allows to incorporate quality of service into telematic applications using a quality of service interceptor (IQoS) and a DiffServ Bandwidth Broker managing a DiffServ domain.
Keywords: DiffServ Architecture, Bandwidth Broker, QoS Broker, CORBA, Linux QoS.

1 Introdução

O perfil de uso da Internet tem mudado drasticamente nos últimos anos. O tipo de terminal utilizado para acesso a Internet, antes limitado aos computadores pessoais, hoje inclui telefones celulares, computadores de mão e aparelhos eletrônicos de áudio e vídeo. O uso de novos tipos

*Aluno de Doutorado da FEEC-UNICAMP.
†Pesquisadora do Centro de Pesquisas Renato Archer - CenPRA.

de dispositivos contribuiu para o aumento de terminais conectados à Internet, aumentando o congestionamento da rede mas também estimulando o desenvolvimento de novos serviços. Podemos apontar as aplicações telemáticas como exemplo de tais serviços. Aplicações telemáticas incluem telefonia sobre IP (voz sobre IP - VoIP), difusão de áudio e vídeo, teleconferência e laboratórios virtuais.

O uso da Internet como rede multiserviços tem influenciado a proposição de arquiteturas e mecanismos para provimento de QoS (Qualidade de Serviço), dado que a Internet foi concebida originalmente para prover serviços que não exigem banda garantida, limites de atraso e *jitter* próximo de zero. Os novos serviços telemáticos fazem uso de fluxos contínuos de áudio e/ou vídeo que dependem de certas garantias de QoS da rede. Podemos classificar as aplicações em dois grandes grupos: Aplicações de tempo-real (serviços telemáticos, por exemplo) e aplicações elásticas. As aplicações de tempo-real podem ser classificadas ainda em não-adaptativas, que são aquelas extremamente sensíveis ao *jitter* e as adaptativas, que são tolerantes ao *jitter*. As aplicações elásticas não fazem uso de fluxos contínuos de áudio e/ou vídeo e não possuem natureza de tempo-real.

Duas arquiteturas para provimento de QoS merecem destaque. A Arquitetura de Serviços Integrados [1] (IntServ) gerencia QoS por fluxo individual e possui controle de admissão em cada roteador. A Arquitetura de Serviços Diferenciados [2] (DiffServ) gerencia QoS por agregados de fluxos e possui um único elemento que faz controle de admissão, o *Bandwidth Broker*.

Este artigo descreve um mecanismo para incorporação de QoS em aplicações telemáticas utilizando como infra-estrutura uma rede dotada de um *Bandwidth Broker* DiffServ e de roteadores com sistema operacional Linux[3, 4, 5] que honram a marcação DiffServ dos pacotes. Um ponto de relevância na nossa implementação é que a negociação de qualidade de serviço entre a aplicação e a rede é transparente para a aplicação através do uso de um interceptador de QoS (IQoS). Também apresentamos uma configuração de rede apropriada para a arquitetura DiffServ. Utilizamos como estudo de caso, para o mecanismo de negociação de QoS com a rede, o REAL [6, 7], um serviço de laboratório virtual acessível através da Internet.

O artigo está dividido em 6 seções subsequentes. A Seção 2 fornece uma breve descrição da arquitetura DiffServ e apresenta os elementos da nossa implementação de forma sucinta. A Seção 3 detalha o mecanismo de requisição de QoS que a aplicação faz uso e apresenta um exemplo de uso no REAL. A Seção 4 detalha a interface do elemento central da arquitetura DiffServ. A Seção 5 detalha a operação dos roteadores e apresenta as disciplinas de fila criadas em cada roteador. A Seção 6 apresenta resultados obtidos com as configurações de rede. Finalmente, a Seção 7 apresenta as conclusões.

2 Arquitetura DiffServ

A utilização da arquitetura IntServ na Internet é inviável, uma vez que tal arquitetura não é escalável. Como a gerência de QoS se dá por fluxo individual em cada roteador, há a necessidade de manter-se uma tabela das conexões em cada roteador do caminho em que o fluxo percorre. Por esta razão utilizamos em nossa implementação a arquitetura DiffServ como mecanismo para priorizar pacotes de determinados fluxos, provendo assim diferenciação de QoS por agregado de fluxos.

A Figura 1 mostra, de forma simplificada, uma das maneiras da arquitetura DiffServ operar. Uma sinalização, por parte de um *bandwidth broker*, configura os roteadores de borda para mar-

car pacotes de determinados fluxos com um certo DSCP (*DiffServ Code Point*, Tabela 1) [8, 9]. Os roteadores de núcleo são configurados (manualmente ou por gerência de rede) para diferenciar os pacotes de acordo com seu DSCP, atribuindo-os a filas que representam serviços DiffServ identificados por um comportamento padrão (*Per Hob Behaviour*, ou PHB). Cinco PHBs são definidos: EF, AF1, AF2, AF3, AF4 e BE[1]. O policiamento pode se dar tanto nos roteadores de borda quanto nos de núcleo. O policiamento visa garantir que pacotes não ultrapassem o limiar contratado e requisitado no momento anterior ao estabelecimento do fluxo. Na Figura 1, apenas os roteadores de borda fazem o policiamento de tráfego. As setas contínuas mais espessas indicam a sinalização de controle entre aplicação e *Bandwidth Broker* e entre *Bandwidth Brokers* de domínios diferentes. As setas tracejadas indicam a sinalização de controle exercida pelo *Bandwidth Broker* nos roteadores de borda que compõem o domínio DiffServ. Por fim, as setas contínuas mais finas indicam o sentido do fluxo de pacotes que terão serviço diferenciado.

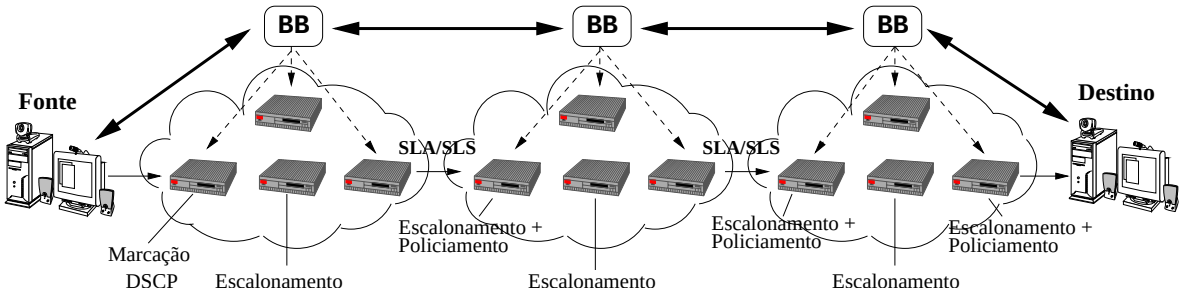

Fig. 1: Arquitetura DiffServ.

Serviço (PHB)	DSCP - Drop Priority 1	DSCP DP2	DSCP DP3
BE	000000		
AF1	001010	001100	001110
AF2	010010	010100	010110
AF3	011010	011100	011110
AF4	100010	100100	100110
EF	101100		

Tab. 1: Mapeamento de PHBs em DSCPs.

Um contrato entre domínios DiffServ, ou SLA (*Service Level Agreement*), estabelece alguns parâmetros de QoS que devem ser respeitados pelo domínio *upstream* (domínio que está injetando pacotes em outro domínio - direção fonte -> destino). O SLA é traduzido para parâmetros de rede como banda, taxa máxima de pacotes, tamanho de rajada, jitter, atraso máximo, etc., em um documento auxiliar conhecido por SLS (*Service Level Specification*). Esses contratos também existem entre um cliente final e a rede ao qual pertence.

Controle de Admissão

Diferentemente da arquitetura IntServ, na arquitetura DiffServ os nós da rede (roteadores) não fazem controle de admissão. Desta forma, é necessário que algum elemento da arquitetura

[1]EF - Expedict Forward (Encaminhamento Expresso); AF - Assured Forward (Encaminhamento Garantido); BE - Best Effort (Melhor Esforço).

o faça. O *Bandwidth Broker* (BB) é responsável em gerenciar a alocação de banda no domínio ao qual pertence, sendo este elemento que deve ser contactado para requisição de banda e determinado serviço (PHB). O BB também é responsável pela negociação entre domínios DiffServ para provimento de qualidade de serviço fim-a-fim.

Componentes do Protótipo Implementado

Os elementos Interceptador de QoS (IQoS), *Bandwidth Broker* (BB), *Daemon* para Controle de Tráfego (DCT) e Interface de Gerência (IG), apresentados na Figura 2, fazem parte do protótipo que implementamos para provimento de serviços diferenciados em um domínio DiffServ. IQoS é o elemento que fica no lado da aplicação responsável por negociar QoS com o BB. DTC, um *daemon* presente em cada roteador de borda, é responsável por sinalizar o núcleo do sistema operacional Linux em função de requisições feitas pelo BB. E, como já mencionado, o BB faz controle de admissão no domínio DiffServ. A Interface de Gerência permite interagir com o *Bandwidth Broker* possibilitando a instalação de SLAs e configuração dos roteadores para marcação de pacotes, inclusive com a possibilidade de instalar filtros permanentes. A comunicação entre IQoS-BB, BB-IG e BB-DTC se dá via CORBA [10]. Apesar de existir o protocolo COPS (*Common Open Policy Service*) [11] para ser utilizado como padrão de comunicação entre o BB e os roteadores (BB e DTC no nosso caso), o COPS não teve a aceitação esperada e suas funcionalidades podem ser facilmente realizadas através de protocolos de *middleware*, razão pela qual utilizamos CORBA. Nas próximas seções descreveremos em detalhes os elementos IQoS, BB e DTC.

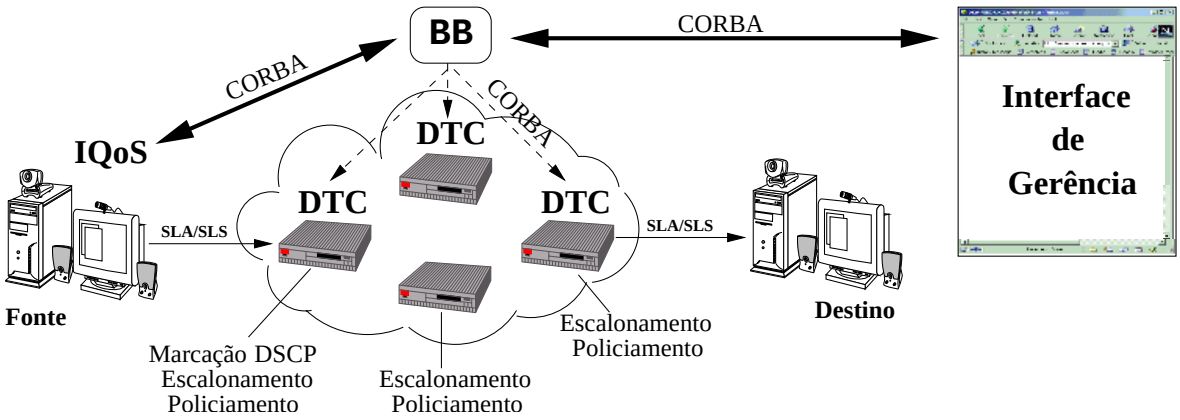

Fig. 2: Distribuição dos elementos da arquitetura.

3 Interceptador de QoS

A fim de liberar o programador de aplicação dos detalhes de comunicação com o BB e da negociação de QoS, e indo ao encontro das tendências em plataformas de *middleware* [12, 13], utilizamos um *Interceptador de QoS* (IQoS) para cuidar de toda a negociação de QoS com a rede. Por utilizarmos CORBA como arquitetura de distribuição de objetos, o IQoS é um mecanismo natural para aumentar a funcionalidade da plataforma de *middleware*, pois passa a fazer parte do ORB após sua inicialização. A função do IQoS é interceptar todas as chamadas de

métodos, filtrando as requisições e dando tratamento adequado a cada uma delas (no caso, contactando o BB quando necessário). IQoS não marca pacotes, tendo como única função contactar o BB. A função de marcação ocorre no núcleo do Linux através das disciplinas de fila. IQoS possui tabelas de mapeamento de QoS de alto-nível para parâmetros de rede (banda). Alguns exemplos de mapeamento de áudio e vídeo podem ser vistos nas Tabelas 2 e 3 respectivamente[2].

audio_ quantisation	**audio_ sampleRate**	**audio_ sampleSize**	**audio_ numChannels**	**Banda** (bps)
uLaw	48000	16	2	152000
uLaw	44100	16	2	150000
uLaw	48000	16	1	75000
uLaw	8000	8	1	74000

Tab. 2: Exemplos de tradução de QoS de áudio de alto nível em banda.

video_ framerate	**video_ colorDepth**	**video_ colorModel**	**video_ resolution**	**Banda** (bps)
30.0	24	JPEG	160x120	280000
30.0	24	JPEG	320x240	400000
7.5	24	JPEG	160x120	70000
7.5	24	JPEG	320x240	100000

Tab. 3: Exemplos de tradução de QoS de vídeo de alto nível em banda.

É importante ressaltar que os valores de banda apresentados nas Tabelas 2 e 3 dependem muito do tipo de codificação de áudio ou vídeo empregada (`audio_quantisation` e `video_colorModel`). Uma solução ideal seria a adaptação da banda durante toda a duração do fluxo de áudio e/ou vídeo [15]. Tomemos como exemplo a codificação MPEG para vídeo. MPEG não gera sempre quadros com a mesma quantidade de bytes, pois utiliza um algoritmo de otimização que permite gerar apenas os fragmentos da imagem que foram alterados aproveitando os quadros anteriores para montar a nova imagem. Portanto, os valores de banda apresentados são utilizados apenas para fins de teste dos mecanismos de alocação/desalocação de banda, sendo necessário outros estudos para chegar a valores otimizados e que possam ser alterados dinamicamente.

Dinâmica de Operação

Para execução dos testes do mecanismo de requisição de serviço ao BB, utilizamos o REAL, um serviço de laboratório virtual acessível através da Internet. Através de um navegador Web capaz de executar applets Java, REAL permite que um usuário opere um robô presente no laboratório de robótica localizado no CenPRA[3]. Duas câmeras de vídeo, uma panorâmica e uma embarcada, captam os movimentos do robô e geram dois fluxos de vídeo. Estes fluxos são transmitidos através da rede e apresentados no navegador do usuário para que o mesmo possa acompanhar a execução da missão. Os pacotes pertencentes aos dois fluxos de vídeo devem

[2]Os nomes dos parâmetros de QoS estão em conformidade com a especificação de Audio/Video Streams do OMG [14].

[3]Centro de Pesquisas Renato Archer (ex-ITI).

ser priorizados em detrimento de outros pacotes que não possuem natureza de tempo-real ou interativa. Para isso, em algum momento anterior ao estabelecimento dos fluxos de vídeo, a rede deve ser sinalizada para que o parâmetro de QoS *banda garantida* possa ser honrado. Como o REAL tem uma natureza interativa, é interessante que os fluxos de vídeo possam ser recebidos pelo usuário tão logo eles estejam disponíveis na rede. Nesse caso, o que está em jogo é a sensação de imersão do usuário no laboratório virtual, que é um dos objetivos do mundo virtual. Quanto menor for o atraso e *jitter* dos fluxos de vídeo, maior a sensação de imersão percebida pelo usuário.

Uma vez terminada a aplicação, a rede deve ser sinalizada para desfazer as "reservas", uma vez que outras requisições serão negadas caso toda a banda passível de reserva já tenha sido alocada.

A infra-estrutura de testes é composta por quatro sub-redes interligadas por quatro roteadores configurados para operar segundo a arquitetura DiffServ. Os pacotes passam por no máximo dois roteadores até atingirem seu destino, por isso não temos a figura única do roteador de borda ou núcleo mas temos algo conjugado (borda+núcleo), o que não compromete a dinâmica de operação desejada. Em cada roteador existe um *daemon*, DTC, responsável em sinalizar o núcleo do Linux em função de requisições de QoS feitas pelo *Bandwidth Broker*. Este *daemon* será detalhado na Seção 5.

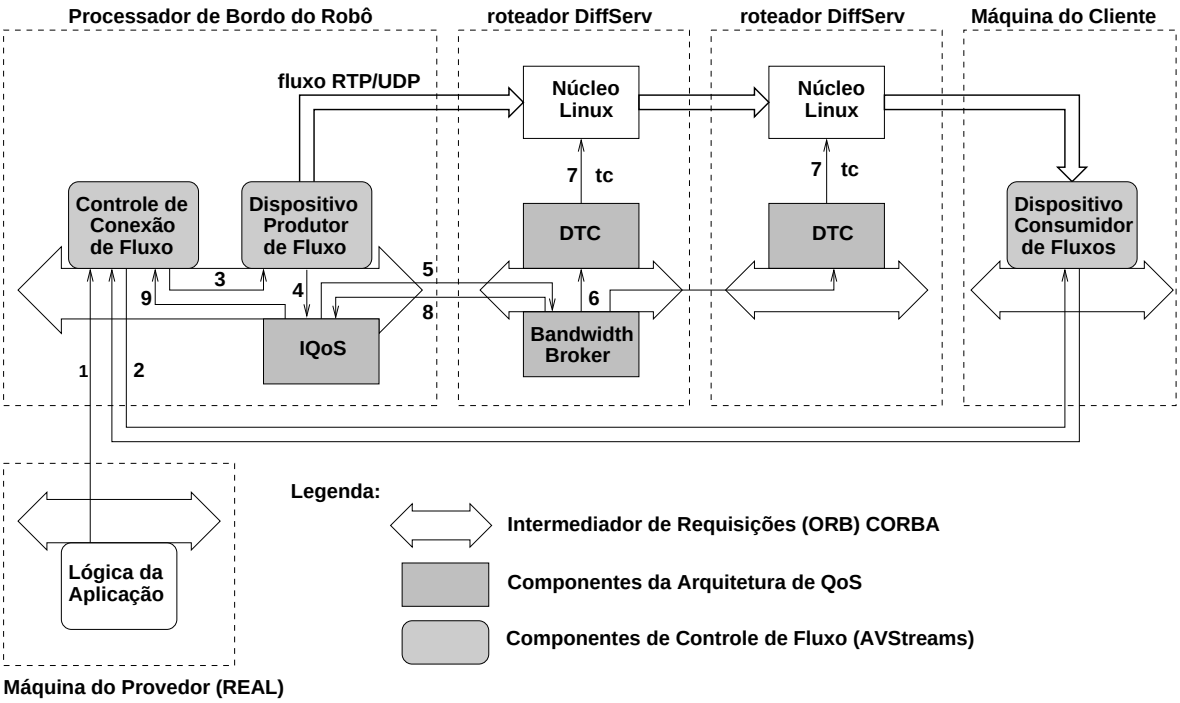

Fig. 3: Dinâmica de operação de reserva de reserva de recursos.

A seguinte seqüência de eventos ocorre no estabelecimento de fluxo de vídeo no REAL (ver Figura 3):

1. A aplicação requisita ao serviço AVStreams uma conexão de fluxo entre um produtor de fluxo e um consumidor de fluxo, fornecendo determinados parâmetros de qualidade de serviço [16];
2. O serviço AVStreams solicita ao consumidor uma porta RTP/UDP para o envio de mídia e confirmação dos parâmetros de QoS (Por exemplo, 30.0 quadros por segundo, 24 bits,

JPEG e resolução de 320x240). O consumidor de fluxo retorna parâmetros de QoS iguais ou inferiores ao solicitado.

3. O serviço AVStreams solicita ao produtor o envio de fluxo para a porta do consumidor com os parâmetros de QoS retornados pelo consumidor. O produtor poderá retornar parâmetros de QoS iguais ou inferiores ao solicitado (por exemplo, 7.5, 24, JPEG, 160x120).
4. IQoS intercepta o retorno desta requisição. Os parâmetros de QoS de alto nível são traduzidos para parâmetros de rede, notadamente banda (Por exemplo, 70000bps) - veja Tabelas 2 e 3;
5. IQoS contacta o BB requisitando a banda estabelecida no passo anterior. BB verifica se a requisição pode ser atendida (autenticação, disponibilidade de recursos, etc.). Caso a requisição não possa ser atendida, execute o passo 9b;
6. BB requisita ao *daemon* DTC para sinalizar a rede;
7. DTC sinaliza o núcleo do Linux para incluir mais um fluxo na lista de fluxos que devem ser tratados com certo QoS;
8. BB retorna um identificador para a requisição (utilizado posteriormente para liberar os recursos associados à requisição);
9. (a) IQoS retorna e a aplicação continua; ou (b) IQoS lança uma exceção, indicando que a reserva não foi possível (cabe ao programador da aplicação tomar medidas posteriores).

Nesta interação, podemos observar dois níveis de negociação de QoS. O primeiro nível ocorre na camada de aplicação (passos 2 e 3). O segundo nível (passos 5 a 9) acontece entre a aplicação (através do IQoS) e um elemento gerenciador de recursos da rede, o *Bandwidth Broker*.

4 O Bandwidth Broker

O BB foi implementado totalmente em Java segundo a arquitetura CORBA, utilizando somente classes e ferramentas da distribuição Java J2SDK [17] versão 1.4.1. O POA (*Portable Object Adapter*) é utilizado como adaptador de objeto. O serviço de nomes CORBA `orbd`, presente na própria distribuição J2SDK, é utilizado para registro do objeto persistente que oferece os serviços do BB. A interface do BB possui os seguintes métodos:

- `installSLA` - permite estabelecer um contrato de serviços no domínio DiffServ;
- `requestQoS` - permite utilizar recursos contratados no domínio DiffServ na forma de reserva de banda;
- `cancelRequest` - permite liberar recursos requisitados através do método `requestQoS`;
- `removeSLA` - permite revogar um contrato feito através do método `installSLA`;
- `listSLA` - permite listar SLAs presentes no BB;
- `listRequests` - permite listar reservas válidas de um determinado usuário;
- `register` - utilizado pelo *daemon* DTC, e somente por este, para registrar-se no BB;
- `unregister` - utilizado pelo *daemon* DTC, e somente por este, para desregistrar-se do BB.

5 O Daemon DTC

Cada roteador possui um *daemon* (DTC), construído em Java, que o BB deve contactar para que os roteadores de borda sejam sinalizados. DTC faz o papel de mediador entre roteador e *Bandwidth Broker*. A sinalização gerada pelo BB acarreta a instalação de filtros para executar a marcação DiffServ (atribui um DSCP ao campo TOS/DS do protocolo IP)[4]. A filtragem é feita segundo endereço IP e protocolo de transporte do destino do fluxo. DTC é capaz de detectar todas as interfaces de rede presentes no roteador e configurá-las para honrar o PHB definido para cada classe de serviço DiffServ[5]. A sinalização entre DTC e o núcleo do Linux se dá via utilitário `tc` (*Traffic Controller*)[18], que permite instalar/desinstalar disciplinas de fila (`qdisc`), classes e filtros de pacotes. Quando DTC é instanciado ele se registra no BB para que possa receber requisições.

Configuração dos Roteadores

O núcleo do sistema operacional Linux honra o campo TOS do cabeçalho IP sem a necessidade de prévia configuração. A disciplina de fila *default* `pfifo_fast` é do tipo FIFO e possui 3 bandas (classes) internas (0,1,2) imutáveis e não configuráveis[6]. A banda 0 é a mais prioritária e enquanto houver pacotes na banda 0, as outras bandas não serão processadas. A referência [18] descreve o mapeamento *default* do campo TOS para as bandas 0, 1 e 2. A disciplina de fila `pfifo_fast` não é suficientemente poderosa para ser utilizada em interfaces de rede de máquinas em um domínio DiffServ. Por isso utilizamos a disciplina de fila `dsmark`, criada especialmente para a arquitetura DiffServ. Deve-se notar que `dsmark` não é a única disciplina de fila que permite criar configurações adequadas para DiffServ.

As disciplinas de fila dos roteadores de núcleo são configuradas manualmente através de *scripts* baseados no utilitário `tc` (estas disciplinas não necessitam ser alteradas com freqüência). A Figura 4 mostra as disciplinas de fila dos roteadores de núcleo. Existem 4 disciplinas de fila do tipo GRED (*Generalized Random Early Detection/Discard*), cada uma configurada com 3 níveis de descarte, apropriadas para o PHB AF; 1 disciplina de fila do tipo `pfifo` para o PHB EF; e 1 disciplina de fila RED (*Random Early Detection/Discard*) para BE. Cada uma das filas do PHB AF possui garantia de 15 Mbits/s, o PHB EF 10 MBits/s e o BE 30 Mbits/s.

Apresentamos a seguir um exemplo de fragmento do *script* de configuração dos roteadores de núcleo para o serviço EF (Figura 4):

```
tc qdisc add dev eth0 handle 1:0 root dsmark indices 64 set_tc_index
...
tc qdisc add dev eth0 parent 1:0 handle 2:0 cbq bandwidth 100Mbit \
    avpkt 1000 cell 8
...
tc class add dev eth0 parent 2:0 classid 2:5 cbq bandwidth 100Mbit \
   rate 10Mbit weight 1Mbit prio 2 allot 1514 maxburst 10 \
   avpkt 1000 bounded
tc qdisc add dev eth0 handle 3:0 parent 2:5 pfifo limit 10
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
   handle 0x2e tcindex classid 1:151
```

[4]TOS/DS - *Type of Service/DiffServ*.
[5]Cada disciplina de fila está associada a uma única interface do roteador.
[6]A não ser pela atribuição de mapeamento dos valores do TOS às 3 bandas.

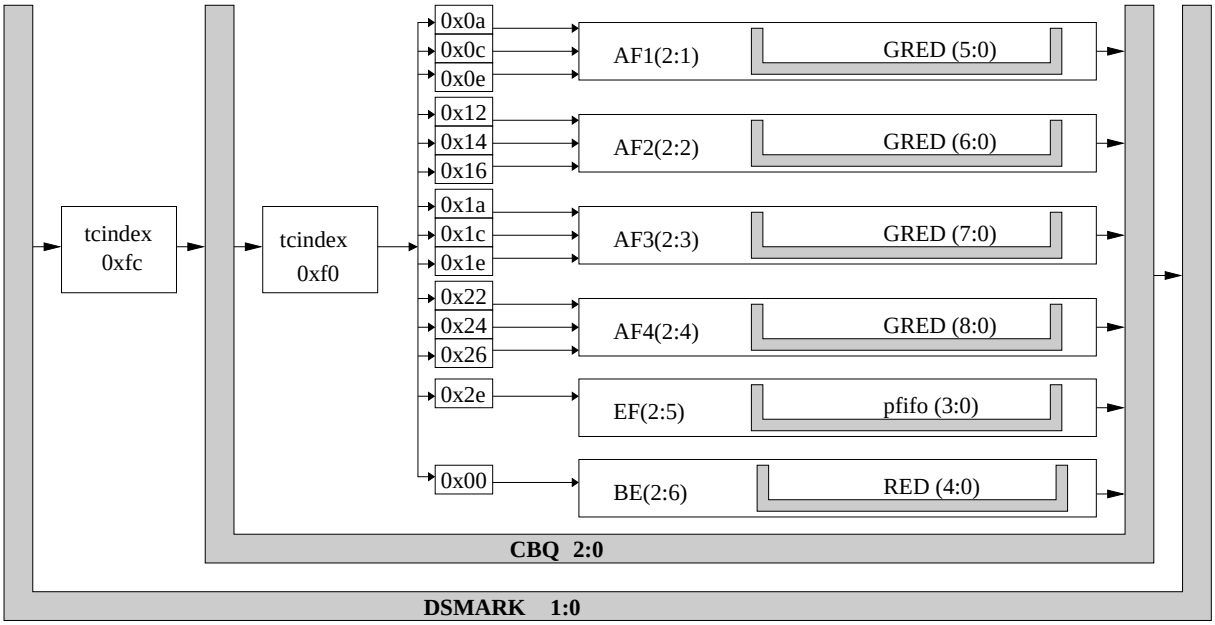

Fig. 4: Configuração de filas nos roteadores de núcleo.

```
tc filter add dev eth0 parent 2:0 protocol ip prio 1 \
   handle 5 tcindex classid 2:5
...
```

Diferentemente dos roteadores de núcleo, as disciplinas de fila dos roteadores de borda são configuradas pelo *Bandwidth Broker*, como visto na Seção 3. São criadas 14 classes que representam cada PHB DiffServ e respectivos níveis de descarte (BE, AF11, ..., AF43 e EF).

A Figura 5 mostra as disciplinas de fila dos roteadores de borda. Existem 2 tipos de disciplinas de fila que utilizamos nos roteadores de borda. No dispositivo de ingresso do roteador (`eth0` na máquina `iracema`, Figura 6) utilizamos a disciplina de fila do tipo INGRESS para proceder a marcação de pacotes e policiamento com um filtro u32 (alterado a cada inclusão/exclusão de fluxos). Este policiamento na borda da rede é importante pois evita que o núcleo da rede fique sobrecarregado. Já o policiamento executado no dispositivo de ingresso tem o papel de economizar tempo do processador, pois descarta os pacotes que excederem a taxa contratada antes mesmo de se proceder o encaminhamento dos mesmos. No dispositivo de egresso (`eth1` na Figura 5 utilizamos `dsmark` para marcação de pacotes. Alguns testes foram conduzidos com relação ao descarte de pacotes nos roteadores de borda, e dispositivo de ingresso, e este mostrou-se extremamente eficiente, sendo preferível do que o descarte de pacotes no núcleo da rede (veja Seção 6).

O próximo exemplo mostra um fragmento do *script* de configuração (gerado pelo DTC e executado a pedido dele) dos roteadores de borda para marcar pacotes da fonte 10.10.3.6 (`palmeiras`) e destino 10.10.1.3 (`joaquina`), protocolo de transporte UDP e porta de destino 5000. Neste exemplo, a aplicação requisitou uma banda de 1Mbit e serviço EF ao *Bandwidth Broker*. Para garantir que a aplicação não exceda a taxa contratada, é feito um policiamento já no ingresso do domínio. A palavra `continue`[7] do filtro indica que se o tráfego exceder o

[7]Existem 3 ações possíveis quando detectada uma sobretaxa de envio - `continue`, `drop` e `reclassify`.

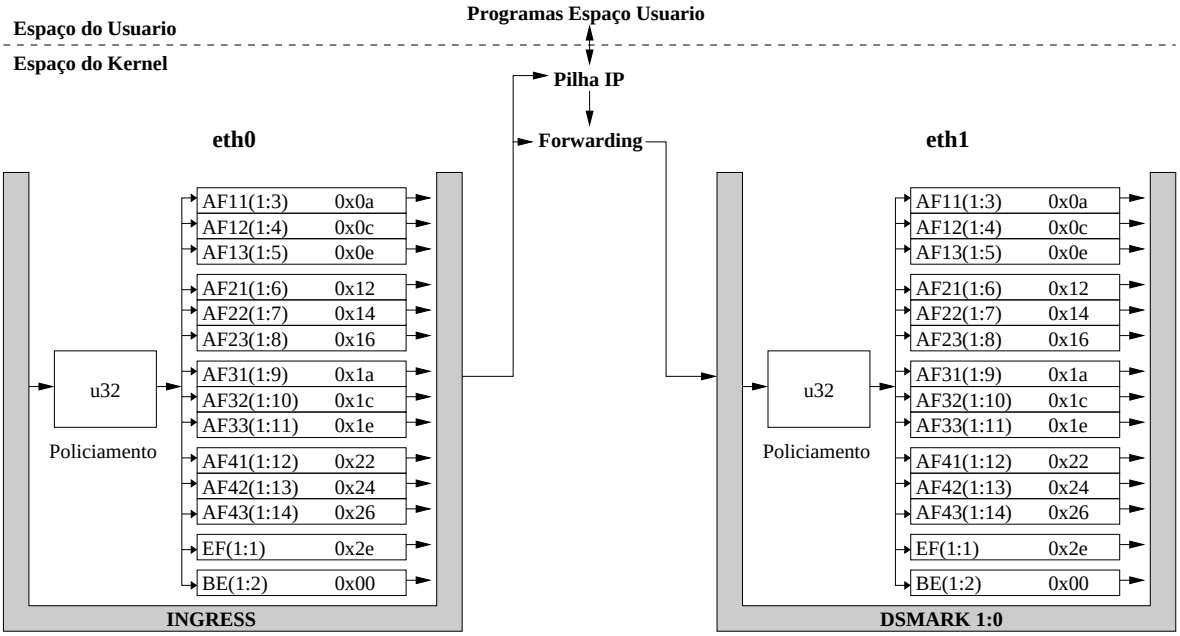

Fig. 5: Configuração de filas nos roteadores de borda.

limite os filtros adiante serão utilizados para um possível casamento (matching). Se nenhum casamento for feito, o pacote é encaminhado com o serviço BE (Melhor-Esforço).

```
tc qdisc add dev eth0 handle ffff: ingress

tc filter add dev eth0 parent ffff: protocol ip prio 1 u32 \
match ip src 10.10.3.6/32 \
match ip dst 10.10.1.3 \
match ip dport 5000 0xffff \
match u8 64 0xff at 8 \
match ip protocol 0x11 0xff \
police rate 1MBit burst 2K \
continue flowid :1
...

tc qdisc add dev eth0 handle 1:0 root dsmark indices 64
...

#EF - TOS (0xb8) - DSCP (0x2e)
tc class change dev eth0 classid 2:1 dsmark mask 0x3 value 0xb8
#BE - TOS 0 (0x00) - DSCP 0 (0x00)
tc class change dev eth0 classid 2:2 dsmark mask 0x3 value 0x00
#AF11 - TOS 40 (0x28) - DSCP 10 (0x0a)
tc class change dev eth0 classid 2:3 dsmark mask 0x3 value 0x28
...
```

6 Testes da Arquitetura

Esta seção apresenta alguns resultados obtidos em testes com os mecanismos de QoS oferecidos pelo núcleo do Linux e valida as configurações utilizadas para criação de disciplinas de fila e classes nos roteadores, notadamente GRED que permite criar disciplinas de fila com vários níveis de descarte (utilizado para implementar o PHB AF).

Todos os roteadores são PCs com processadores Pentium III com freqüência de 1GHz e sistema operacional Linux (núcleo 2.4.7-10 e distribuição RedHat 7.2)[8] possuindo interfaces de rede do tipo Fast Ethernet que operam a 100Mbits/s.

Para os testes criamos nos roteadores de núcleo 4 disciplinas de fila do tipo GRED, cada uma configurada com 3 níveis de descarte, utilizadas nos quatro PHBs AF; 1 disciplina de fila do tipo RED para ser utilizada como PHB BE; e 1 disciplina de fila do tipo `pfifo` para ser utilizada como PHB EF.

Os serviços (PHB) foram configurados conforme a Tabela 4. Estas configurações limitantes foram feitas apenas para fins de teste, as configurações utilizadas normalmente foram descritas na Seção 5.

PHB	Banda(Mbps)	`bounded`	`isolated`	Descartado
AF11	15	Sim	Sim	Terceiro
AF12				Segundo
AF13				Primeiro
AF21	3	Não	Não	Terceiro
AF22				Segundo
AF23				Primeiro
AF31	15	Sim	Sim	Terceiro
AF32				Segundo
AF33				Primeiro
AF41	15	Sim	Sim	Terceiro
AF42				Segundo
AF43				Primeiro
EF	5.8	Não	Não	N/A
BE	46.2	Sim	Sim	N/A

Tab. 4: Banda reservada para cada PHB.

Na Tabela 4, o termo *isolated* significa que a banda reservada não pode ser emprestada caso não esteja sendo utilizada e o termo *bounded* significa que uma determinada classe não emprestará banda de outra classe caso necessite. Em outras palavras, somente as disciplinas de fila dos serviços AF2 e EF poderão emprestar ou ter sua banda emprestada caso haja necessidade.

Neste primeiro teste apresentado, os roteadores de borda marcam os pacotes de 3 fluxos gerados pelo utilitário MGEN [19]. Nenhum policiamento com relação a sobretaxa de envio de pacotes é feito (os filtros são configurados para deixar passar até 100Mbit/s, que é a taxa máxima do enlace).

A Figura 6 apresenta a rede de testes utilizada. A máquina `palmeiras` é utilizada como fonte geradora de tráfego enquanto a máquina `joaquina` é utilizada como consumidora. A

[8] Esta distribuição do Linux oferece o núcleo do sistema já compilado com todos os recursos de QoS necessários para os experimentos.

máquina `iracema` é o roteador de ingresso enquanto a máquina `trindade` faz o papel de roteador de núcleo e de egresso.

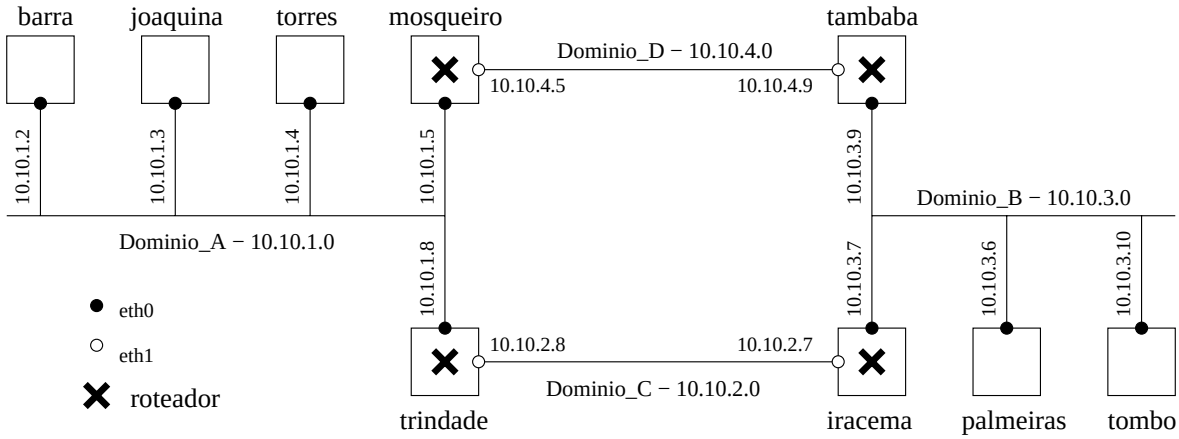

Fig. 6: Rede de testes.

Os fluxos são gerados pelo utilitário MGEN nas seguintes taxas e são marcados pelo roteador de borda com os DSCPs dos seguintes serviços (PHBs): 3Mbps e PHB AF21; 1Mbps e PHB AF22; e 5.5Mbps e PHB EF. O fluxo com serviço EF inicia em t40 e termina em t70 (Figura 7). Enquanto a banda para o serviço EF está desocupada, o serviço AF2 empresta de EF 1Mbit/s para poder utilizar 4Mbit/s, pois a banda reservada para AF2 é de apenas 3Mbit/s. Quando o roteador `trindade` recebe um fluxo com serviço EF no tempo t40, são consumidos 5.5Mbit/s da banda de EF, restando apenas 0.3Mbit/s para emprestar. Neste ponto, alguns pacotes do serviço AF22 são descartados, deixando passar apenas 0.3Mbit/s. Podemos observar, ainda na Figura 7, que a configuração aplicada funcionou conforme o esperado, pois o serviço AF22 teve seus pacotes descartados antes dos pacotes do serviço AF21, justamente o comportamente esperado. Após o término do uso da banda do serviço EF no tempo t70, o serviço AF22 passa a emprestar de EF 1Mbit/s novamente.

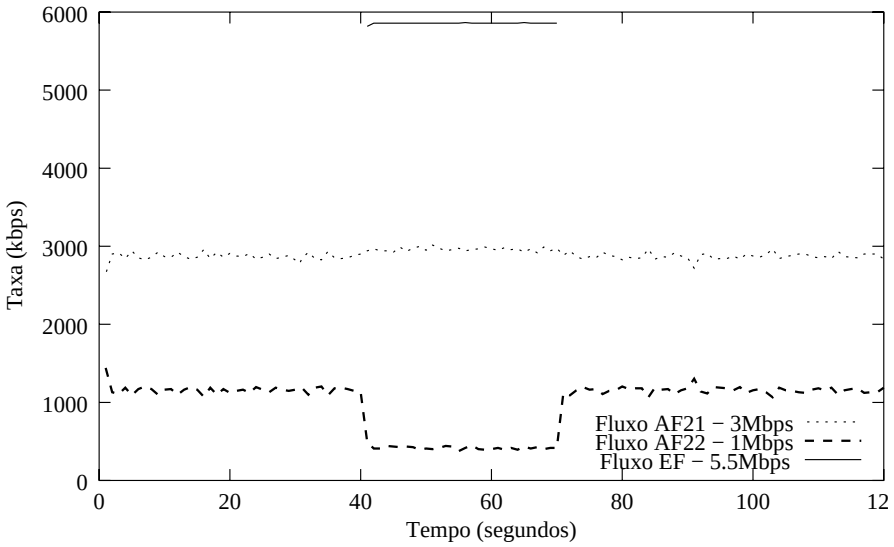

Fig. 7: Três fluxos policiados somente pelos roteadores de núcleo.

Apresentamos agora o segundo teste, onde os roteadores de borda fazem o policiamento já no dispositivo de ingresso. Mantivemos o mesmo *script* de configuração do primeiro teste apresentado, ou seja, o MGEN gera 3 fluxos com taxas 5.5Mbit/s, 3Mbit/s e 1Mbit/s. No roteador de ingresso (`iracema`) limitamos a taxa do fluxo com serviço EF de 5.5Mbit/s para 1Mbit/s. A Figura 8 mostra as taxas de pacote dos 3 fluxos recebidos pela máquina `joaquina` (10.10.1.3). Até o tempo t50 nenhum policiamento foi feito na borda. A partir do tempo t51 o policiamento passou a ser feito em `iracema` (roteador de ingresso), descartando pacotes e reduzindo a taxa do fluxo EF para 1Mbit/s. Observe que, a partir do tempo t51, o núcleo da rede passou a enviar o fluxo AF22 novamente com taxa de 1Mbit/s pois a banda destinada a EF não está totalmente em uso.

Este mesmo tipo de policiamento é feito quando o BB configura os roteadores de borda para marcar pacotes de determinados fluxos, deixando passar somente a taxa que o usuário requisitou inicialmente.

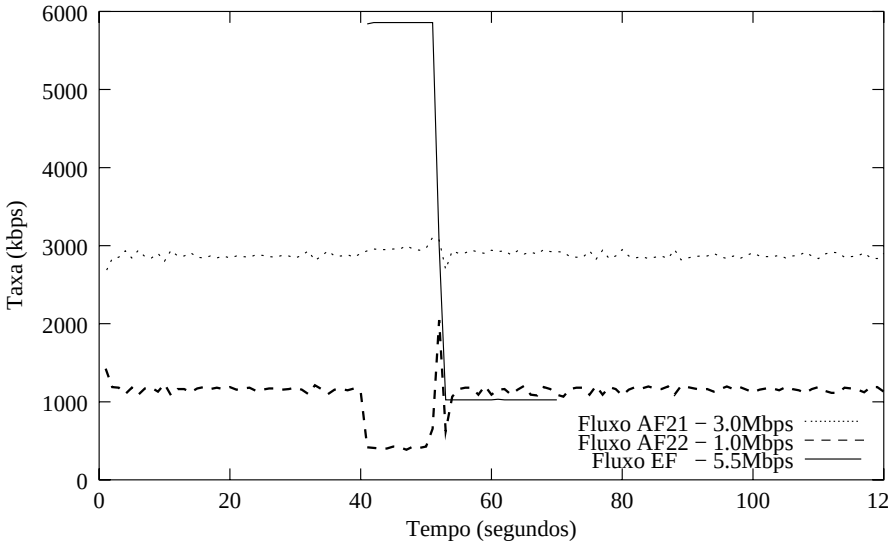

Fig. 8: Três fluxos policiados pelos roteadores de borda.

7 Conclusões

A necessidade de arquiteturas de QoS para a Internet se mostra evidente em um contexto em que esta rede vem sendo utilizada como rede multiserviço. Dentre as arquiteturas existentes, a que tem se mostrado mais promissora é a Arquitetura de Serviços Diferenciados (DiffServ).

Neste artigo apresentamos a implementação de um *Bandwidth Broker DiffServ* como suporte para o provimento de QoS em aplicações telemáticas. Um ponto de relevância na nossa implementação, e não visto em outras publicações, é a interação entre a aplicação que provê o serviço telemático e o BB, ocorrendo com total transparência uma vez que o próprio *middleware* se encarrega da negociação de QoS através do Interceptador de QoS (IQoS). O núcleo do Linux provê todos os elementos necessários para o provimento de qualidade de serviço em uma rede. Em particular, as disciplinas de fila `dsmark` e `GRED` são importantes, pois permitem configurar o núcleo do Linux para implementar os serviços previstos na arquitetura DiffServ (AF, EF e BE). Os resultados dos testes apresentados mostraram que as configurações criadas em nossa rede de testes, através de *scripts* do utilitário `tc`, funcionam corretamente.

Trabalhos Relacionados

Existem algumas implementações de *Bandwidth Brokers* DiffServ desenvolvidas por outros grupos de pesquisa, talvez as mais próximas da nossa implementação sejam:

- KUBB [20]. O KUBB, da Universidade de Kansas, foi desenvolvido para roteadores Cisco e Linux e a marcação de pacotes pode ser feita no roteador de ingresso ou na própria máquina geradora de fluxo. Para isso, existem *daemons* instanciados na máquina cliente e em cada roteador de borda do domínio DiffServ. Existe uma interface de gerência capaz de configurar o BB mas não possui um mecanismo para que a própria aplicação possa requisitar um determinado serviço.
- Multi-Layer Bandwidth Broker [21]. A arquitetura deste bandwidth broker é composta por uma camada de controle conhecida por RCL (*Resource Control Layer*), a qual possui duas sub-camadas. Uma das sub-camadas é responsável pelo controle de admissão e a outra pela gerência de rede. A arquitetura também provê um elemento chamado AMW (*Application Middleware*) que permite as aplicações requisitarem QoS ao BB. O mecanismo pelo qual as requisições são feitas não é especificado, portanto não é possível saber se as requisições são transparentes ao programador ou se existem métodos para serem chamados explicitamente. Os elementos da arquitetura são implementados em Java como objetos CORBA.

Trabalhos Futuros

Podemos identificar os seguintes trabalhos futuros:

- Explorar a negociação entre domínios DiffServ. Essa negocição mostra-se essencial para o provimento de qualidade de serviço fim-a-fim, uma vez que os pacotes de determinado fluxo podem passar por mais de um domínio DiffServ. Apesar de existirem algumas soluções propostas por alguns grupos de pesquisa, não existe um padrão definido de negociação entre domínios DiffServ, padrão este que deve ser definido com certa urgência, uma vez que a Internet 2 deverá utilizar a arquitetura DiffServ para provimento de QoS;
- Incorporar mecanismos de gerência de tráfego na arquitetura de QoS utilizando redes MPLS (utilizaremos uma implementação desenvolvida pelo próprio grupo [22, 23]). Na nossa implementação atual, não é possível garantir que um determinado enlace dentro do domínio DiffServ comportará todos os fluxos operando nos respectivos serviços (PHBs), isso porque não influimos no roteamento dos pacotes. Todo o roteamento é feito pelos mecanismos naturais do protocolo IP. Uma solução ideal seria a utilização do MPLS para criar caminhos comutados e assim garantir banda e determinado serviço em cada um deles. Uma outra solução possível seria a utilização de informação de tráfego em cada roteador, mas, além de ser mais difícil de ser implementada, acreditamos ser menos eficiente que o uso de MPLS;
- Adaptação de banda durante toda a duração de um fluxo de áudio e/ou vídeo. Pretende-se com isso chegar a valores otimizados que possam ser alterados dinamicamente, para mais ou para menos banda, ao longo de uma transmissão de áudio e/ou vídeo. Na corrente implementação não é possível mudar a banda reservada durante a transmissão do fluxo, a não ser pela intervenção manual por gerência de rede, pela interface de gerência (Figura 2), ou ainda pelo próprio programador de aplicação através de chamada explícita dos

métodos do *Bandwidth Broker*. Talvez a utilização de agentes móveis mostre-se interessante para este tipo de tarefa.

Agradecimentos

Esta pesquisa é suportada pelo CNPq (proc. 141880/01) em forma de atribuição de bolsa de doutorado ao primeiro autor e pelo projeto QUARESMA (Edital Redes Avançadas - Chamada CNPq 10/2001).

Referências

[1] R. Braden et. al. “Integrated Services in the Internet Architecture: an Overview”. RFC 1633, Internet Engineering Task Force, June 1994. `http://www.ietf.org/rfc/rfc1633.txt`.

[2] S. Blake et. al. “An Architecture for Differentiated Services”. RFC 2475, Internet Engineering Task Force, December 1998. `http://www.ietf.org/rfc/rfc2475.txt`.

[3] W. Almesberger, J. Salim, A. Kuznetsov. “Differentiated Services on Linux”. Technical report, EPFL ICA, CTL Nortel Networks, INR Moscow, June 1999.

[4] W. Almesberger. “Linux Network Traffic Control - Implementation Overview”. Technical report, EPFL ICA, April 1999.

[5] T. Einsiedler, H. Einsiedler, M. Scheidegger, G. Stattenberger, K. Jonas, H. Stuttgen. “A Linux Implementation of a Differentiated Services Router”. In *INTERWORKING*, pages 302–315, URL ”citeseer.nj.nec.com/326874.html”, 2000.

[6] E. Guimarães, E. Cardozo, M. Magalhães, M. Bergerman, A. Maffeis, J. Pereira, B. Russo, C. Miglinsk, R. Pinto. “Desenvolvimento de Software Orientado a Componentes para Novos Serviços de Telecomunicações”. In *19 Simpósio Brasileiro de Redes de Computadores*, 2001.

[7] E. Guimarães, E. Cardozo, M. Magalhães, M. Bergerman, A. Maffeis, B. Russo, J. Pereira . “REAL: A Virtual Laboratory for Mobile Robot Experiments”. In *Conference Telematics Application in Automation and Robotics - IFAC-TA 2001*, Weingarten, Germany, Jul. 2001.

[8] J. Heinanen et. al. “Assured Forwarding PHB Group”. RFC 2597, Internet Engineering Task Force, June 1999. `http://www.ietf.org/rfc/rfc2597.txt`.

[9] V. Jacobson et. al. “An Expedited Forwarding PHB”. RFC 2598, Internet Engineering Task Force, June 1999. `http://www.ietf.org/rfc/rfc2598.txt`.

[10] OMG. “The Common Object Request Broker: Architecture and Specification – Version 2.3.1”. Document formal/99-10-07, Object Management Group, Outubro 1999. `http://www.omg.org`.

[11] E. Durham et. al. “The COPS (Common Open Policy Service) Protocol”. RFC 2748, Internet Engineering Task Force, January 2000. `http://www.ietf.org/rfc/rfc2748.txt`.

[12] A. Tripathi. “Challenges Designing Next-Generation Middleware Systems”. *Communications of The ACM*, 45(6):39–42, June 2002.

[13] F. Kon, F. Costa, G. Blair, R. Campbell. “The Case for Reflective Middleware”. *Communications of The ACM*, 45(6):33–38, June 2002.

[14] OMG. “The Audio/Video Streams Specification – Version 1.0”. Document formal/00-01-03, Object Management Group, January 2000. `http://www.omg.org`.

[15] C. Koliver, J. Farines. “Um Controlador Nebuloso para Adaptação de QoS”. In *19 Simpósio Brasileiro de Redes de Computadores - SBRC’2001*, Florianópolis, S.C., Maio 2001.

[16] E. Guimarães, E. Cardozo, M. Magalhães, M. Bergerman. “Um Framework de Transmissão de Áudio e Vídeo para Novos Serviços de Telecomunicações”. In *18 Simpósio Brasileiro de Redes de Computadores*, 2000.

[17] Sun Microsystems. *“Java™ 2 Platform, Standard Edition (J2SE™)”*, 2002. `http://java.sun.com/j2se/`.

[18] B. Hubert, et. al. *“Linux Advanced Routing & Traffic Control HOWTO”*. Nether Labs, Netherlands, September 2002. Open Publication License v1.0.

[19] *“MGEN-3.2 User’s Guide”*. URL ”http://manimac.itd.nrl.navy.mil/MGEN/”.

[20] D. Rao, H. Sitaraman. “Bandwidth Broker Implementation”. Technical report, University of Kansas Information and Telecommunication Technology Center (ITTC), URL: www.ittc.ukans.edu/ krdao/845/, 1999.

[21] G. Politis, P. Sampatakos, I. Venieris. “Design of a Multi-layer Bandwidth Broker Architecture”. Technical report, National Technical University of Athens, Greece. URL: http://www.ntua.gr/en_index.html, 1999.

[22] T. Badan, R. Prado, E. Zagari, E. Cardozo, M. Magalhães. “Uma Implementação do MPLS para Redes Linux”. In *19 Simpósio Brasileiro de Redes de Computadores - SBRC’2001*, Florianópolis, S.C., Maio 2001.

[23] E. Zagari, T. Badan, R. Prado, E. Cardozo, M. Magalhães. “Uma Plataforma para Engenharia de Tráfego com Qualidade de Serviço em Redes MPLS”. In *20 Simpósio Brasileiro de Redes de Computadores - SBRC’2002*, Búzios, R.J., Maio 2002.