Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

Departamento de Engenharia de Computação e Automação Industrial

Dissertação de Mestrado

MPLS–DS: Uma plataforma para validação de políticas no contexto das redes MPLS/DiffServ

Autor: Alex Marcelo Samaniego Guillén

Orientador: Prof. Dr. Mauricio Ferreira Magalhães

Dissertação apressentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, como parte dos requisitos para obtenção do Título de Mestre em Engenharia Elétrica.

Banca Examinadora:
Prof. Dr. Mauricio Ferreira Magalhães (FEEC/UNICAMP)
Prof. Dr. Edmundo Roberto Mauro Madeira (IC/UNICAMP)
Prof. Dr. Leonardo de Souza Mendes (FEEC/UNICAMP)
Prof. Dr. Walter da Cunha Borelli (FEEC/UNICAMP)

Dezembro de 2001

Resumo

A plataforma MPLS−DS tem o objetivo de permitir a validação de políticas de configuração de uma determinada rede através de simulações, para que essas políticas possam ser empregadas posteriormente em uma rede real. Para executar esta tarefa, a plataforma está formada por uma rede MPLS com suporte aos serviços diferenciados, um *bandwidth broker*, uma arquitetura de políticas simplificada (PDP, PEP e repositório de políticas), um mecanismo para o mapeamento das classes de serviço oferecidas dentro do domínio da plataforma e os contratos de serviços estabelecidos com redes clientes. A plataforma utiliza também a arquitetura CORBA para a comunicação do *bandwidth broker* com o PEP de forma a isolar a implementação do *bandwidth broker* da rede simulada. A automatização da configuração dos recursos da rede é realizada por meio do *bandwidth broker* e do PDP baseando-se nas políticas implementadas para o domínio e nos contratos de serviços firmados entre os clientes cujas aplicações estejam associadas a uma determinada classe de serviço. As simulações são realizadas no simulador *NS*.

Abstract

The goal of the MPLS−DS platform is to validate network configuration policies through simulations in order to implement those policies in a real network environment. To achieve this objective, the following components are used in the platform: the MPLS technology with differentiated services support, a Bandwidth Broker, a simplified policy architecture (PDP, PEP, policy repository), a class of services mapping device which is provided within the platform domain and the established service agreements with the client networks. This implementation uses CORBA as the communication platform between the bandwidth broker and the PEPs in order to isolate the bandwidth broker implementation from the simulated network. The PDP and the bandwidth broker are responsible for the automatization of the network resources configuration process. It is based on the policies specified in the network domain and also in the services agreements with the application belonging to a specific class of service. The simulations were accomplished using the NS simulation tool.

Agradecimentos

- Em primeiro lugar quero agradecer a Deus que sempre vela por mim e me da forças para continuar neste caminho.
- Agradeço especialmente a meu orientador Prof. Dr. Mauricio Ferreira Magalhães por sua excelente orientação e por me brindar seu apoio durante todo este tempo.
- A meus avós Julio e Angelica por estarem aqui comigo e a toda minha familia por sempre me apoiar e dar forças para concluir esta dissertação. A Seu Mário e Dona Mercedes por me brindar seu carinho e por me considerar um neto para eles.
- A meus tios Erly e Neldy por me hospedar e me dar toda a liberdade para permanecer aqui.
- Aos amigos que conheci nesta faculdade, por me ensinar tanto na parte acadêmica como pessoal e também por compartilhar nos momentos de descontração.
- Ao grupo de trabalho (Leo, Nicola, Mabia, Marco, Ricardo, Rodrigo, Tomas, Tulius e Vinicius) pela convivência e ajuda que me deram.
- Ao DCA por me conceder o espaço de trabalho.
- Ao CNPq por me dar o apoio financeiro.

Para mis padres Marcelo y Angela,
mis hermanos Lalo, Delia y Marga,
y mis sobrinas Carito y Angelita.

Sumário

Lista de Figuras

Lista de Tabelas

Acrônimos

AF	Assured Forwarding
API	Application Programming Interface
BA	Behavior Aggregate
CBR	Constraint Based Routing
CLI	Command Line Interpreter
COPS	Common Open Policy Service
CORBA	Common Object Request Broker Architecture
DARPA	Defense Advanced Research Projects Agency
DiffServ	Serviços Diferenciados
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Interface
EF	Expedited Forwarding
FEC	Forwarding Equivalence Class
FIFO	First In First Out
FTN	FEC to NHLFE Map
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
ILM	Incoming Label Map

LBNL	Lawrence Berkeley National Laboratory
LDAP	Lightweight Directory Access Protocol
LDP	Label Distribution Protocol
LSP	Label Switched Path
LSR	Label Switching Router
MATE	MPLS Adaptive Traffic Engineering
MIRA	Minimal Interference Routing Algorithm
MPLS	MultiProtocol Label Switching
NS	Network Simulator
OTCL	Object Tool Command Language
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PHB	Per-Hop Behavior
PSC	PHB Scheduling Class
PVC	Permanent Virtual Connection
QoS	Quality of Service
RAR	Resource Allocation Requests
RED	Random Early Detection
RFC	Request for Comments
RIO	RED In Out
RIO-C	RIO Coupled
RIO-D	RIO De-coupled
RSVP	Resource Reservation Protocol
SLA	Service Level Agreement

SLS	Service Level Specification
SNMP	Simple Network Management Protocol
TCA	Traffic Conditioning Agrement
TCP	Transmission Control Protocol
TEQUILA	Traffic Engineering for Quality of Service in the Internet, at Large Scale
TTL	Time To Live
UCB	University of California Berkeley
UDP	User Datagram Protocol
USC/ISI	University of Southern California/Information Sciences Institute
VINT	Virtual InterNetwork Testbed
WRED	Weighted RED
Xerox PARC	Xerox Palo Alto Research Center

Capítulo 1

Introdução

A rede Internet, atualmente, é um dos meios de comunicação mais importantes a nível mundial utilizada em colégios, universidades, centros de pesquisa, empresas, lares e até em cabines públicas Internet. O seu crescimento, tanto em número de usuários como na variedade de aplicações que a utilizam como infra-estrutura de comunicação, tende a torná-la uma rede multiserviços. Por outro lado, a presença global da Internet a torna atrativa como uma infra-estrutura de comunicação universal para negócios e lazer.

Por isso, é necessário que sejam desenvolvidas formas para que a rede Internet possa atender a maioria de usuários (levando em conta suas necessidades) e aplicações (considerando as suas características). Portanto, segundo [45], a tendência para o *backbone* Internet é estar provido com enlaces de alta velocidade, QoS, engenharia de tráfego, segurança e comutação óptica.

- **Enlaces de alta velocidade**, prover a Internet com enlaces de alta velocidade se torna necessário devido ao crescimento do tráfego que transita pela rede, mas devemos levar em conta que isto não resolve a garantia que usuários e aplicações necessitam, pois sempre surgirão novos usuários e aplicações que consigam saturar os enlaces.
- **QoS**, a qualidade de serviço leva a que o clássico modelo de serviço de melhor esforço seja mudado para um modelo que ofereça diferentes níveis de serviços com requisitos específicos de qualidade de serviço. Além disso, novos métodos de gerenciamento dos recursos de rede devem ser fornecidos para auxiliar na configuração da rede, de maneira a automatizar este processo.
- **Engenharia de Tráfego**, a engenharia de tráfego provê a otimização do desempenho da rede e da utilização dos recursos. Por meio da utilização de novas tecnologias, como MPLS,

pode-se criar caminhos na rede considerando-se o roteamento baseado em restrições (CBR) como mecanismo para prover à rede Internet os recursos da engenharia de tráfego.

- **Segurança**, a segurança é necessária pois cada vez mais estão sendo transmitidos dados confidenciais através da Internet, portanto, deve-se prover à rede mecanismos para codificação/decodificação de dados, filtragem de pacotes, autenticação de usuários e autorização para a utilização de recursos.
- **Comutação óptica**, a comutação óptica visa criar um canal, com a ajuda do MPLS, desde a origem até o destino, no qual não seja necessário executar a transformação de sinal óptico para sinal elétrico em cada roteador IP.

A qualidade de serviço pode ser provida por meio da utilização de modelos de serviço tais como os serviços integrados (IntServ) ou serviços diferenciados (DiffServ). O modelo IntServ deixa de ser uma boa opção para ser implementado no núcleo da rede Internet devido a que este realiza reserva de recursos e guarda informações de estados para cada fluxo que ingressa na rede, tornando-o não escalável [11, 45]. Já o modelo DiffServ se torna atrativo pelo fato de que este realiza a diferenciação de classes de serviço para agregações de fluxos [5, 18, 11]. O gerenciamento de recursos de rede pode ser realizado manualmente, pelo administrador da rede, ou pode ser executado automaticamente, por meio dos *bandwidth brokers* em conjunto com uma arquitetura de políticas, como é proposto nesta dissertação nos próximos capítulos.

A engenharia de tráfego pode ser realizada através da utilização do ATM, já que os comutadores ATM são rápidos e possuem a habilidade de criar caminhos virtuais permanentes na malha ATM. Mas, devido à segmentação dos pacotes em células (realizada no comutador ATM de ingresso), a remontagem das células em pacotes (realizada no comutador ATM de egresso), ao alto *overhead* no cabeçalho ATM e devido a que roteadores IP atuais são tão rápidos como os comutadores ATM, fazem com que aumente o interesse pelo uso de outras tecnologias no *backbone* da rede. As redes MPLS podem ser empregadas para prover a engenharia de tráfego já que esta tecnologia separa o roteamento do encaminhamento, permitindo criar caminhos alternativos dentro da rede, chamados de LSPs.

A plataforma MPLS−DS é uma infra-estrutura formada basicamente por duas tecnologias: a comutação baseada em rótulos (MPLS *MultiProtocol Label Switching*) e a diferenciação de serviços (DS ou DiffServ), possui uma arquitetura de políticas e um *bandwidth broker*. Nosso objetivo é, baseado nesta plataforma, prover ao *backbone* Internet as habilidades de diferenciar serviços e aplicar a engenharia de tráfego por meio da utilização das redes MPLS com suporte à diferenciação

de serviços. Além disso, a plataforma possui um sistema de gerenciamento de recursos de rede baseado em um *bandwidth broker* que interage com uma arquitetura de políticas. Nosso interesse é utilizar a plataforma MPLS−DS como uma ferramenta para a validação de políticas de configuração de rede, por exemplo, como configurar a fila para uma determinada classe de serviço ou a que LSP deve estar associado um fluxo de dados que requer uma determinada classe de serviço. A validação de políticas é executada através de simulações de redes realizadas no *NS*[1] [14]. O principal benefício desta implementação é poder utilizar as políticas adotadas nas simulações em um contexto próximo ao de uma rede real.

A dissertação encontra-se estruturada da seguinte forma: O Capítulo 2 é abordado como parte teórica introdutória aos temas de QoS e engenharia de tráfego para a rede Internet. O terceiro e quarto capítulos estão relacionados à proposta desta dissertação e sua aplicação, respectivamente.

No Capítulo 3, apresentaremos a plataforma MPLS−DS, implementada no simulador *NS*, descreveremos suas características, seus componentes e a maneira de como foi implementada. No Capítulo 4, mostramos detalhadamente como a plataforma pode ser utilizada e descrevemos que políticas de configuração foram implementadas para realizar a configuração da rede simulada de forma automática. No Capítulo 5, apresentaremos as conclusões obtidas com a utilização da plataforma e descreveremos os possíveis trabalhos futuros que podem ser realizados dentro deste contexto.

[1] *NS*, Network Simulator.

Capítulo 2

Qualidade de Serviço e Engenharia de Tráfego na Internet

Este capítulo tem como objetivo apresentar uma breve introdução aos temas relacionados à qualidade de serviço e a engenharia de tráfego no contexto da rede Internet e como podem ser providas essas características no núcleo (*backbone*) da rede Internet.

2.1 Qualidade de Serviço na Internet

Como prover *Qualidade de Serviço* (QoS[1]) à rede Internet é um dos temas de pesquisa atualmente realizados em centros universitários, fóruns como o IETF (*Internet Engineering Task Force*) [19], e em grandes projetos como Internet2 [20], TEQUILA [2] [38], entre outros. Abordaremos a QoS do ponto de vista da rede Internet.

2.1.1 O que é QoS?

No contexto de redes de computadores, QoS significa dar diferente tratamento aos pacotes de aplicações distintas e garantir que suas características sejam respeitadas (dependendo do tipo da aplicação). Para poder realizar a diferenciação, os elementos de rede (terminais ou roteadores) devem possuir mecanismos para que as requisições por algum serviço possam ser satisfeitas. A QoS pode ser medida pela latência dos pacotes (atraso), pela variação do atraso dos pacotes (*jitter*), pela taxa de perda de pacotes ou pela vazão total da aplicação. Por exemplo, as aplicações em

[1] QoS, do inglês *Quality of Service*.
[2] *TEQUILA, Traffic Engineering for Quality of Service in the Internet at Large Scale*

tempo real, comércio eletrônico e correio eletrônico possuem características particulares, portanto os pacotes destas aplicações devem ser tratados diferentemente pelos elementos de rede e devem estar associadas a uma classe de serviço. Uma visão geral do significado de QoS é encontrado em [18]. Em [32] e [42] se discute e promove a adoção da QoS para a rede Internet.

Modelos de serviço para a Internet

A IETF tem proposto vários modelos de serviço e protocolos para prover QoS na rede Internet. Entre eles se destacam os modelos dos *Serviços Integrados* (IntServ) e *Serviços Diferenciados* (DiffServ). Em [45] é descrito que os modelos de serviço não são suficientes para prover a QoS na Internet, outras providências devem ser tomadas ou mesmo necessárias, como por exemplo, gerenciamento de perda de pacotes e prevenção de congestionamentos (engenharia de tráfego). Em [25] são realizados testes, através de simulações, sobre a utilização dos serviços diferenciados no *backbone* Internet.

Na continuação, descreveremos brevemente como o modelo IntServ provê QoS à rede Internet. Na Seção 2.1.3, descreveremos detalhadamente o modelo DiffServ.

2.1.2 Serviços Integrados

O modelo de serviço IntServ provê duas classes de serviço: a) o *serviço garantido*, que está associado a aplicações que requerem atraso fixo e b) o *serviço de carga controlada*, associado a aplicações que requerem confiabilidade. Este modelo atua em conjunto com o protocolo RSVP (*Resource Reservation Protocol*). Cada fluxo de pacotes é associado a uma classe de serviço e uma reserva de recursos de rede é feita para o fluxo através do RSVP. Isto demanda que, para cada fluxo, exista no roteador informações de estado, além de se ter implementadas as funções para a classificação, o controle de admissão e o escalonamento de pacotes.

Os problemas com a arquitetura IntServ são:

- A quantidade de estados de informação cresce à medida que os fluxos aumentam na rede, portanto esta arquitetura não é escalável.
- Todos os roteadores devem suportar o protocolo RSVP, controle de admissão, classificação baseada em múltiplos campos e o escalonamento de pacotes.

Detalhes da arquitetura IntServ são encontrados em [33].

2.1.3 Serviços Diferenciados

O modelo de Serviços Diferenciados surgiu devido ao problema de escalabilidade do modelo IntServ no núcleo da rede Internet. O modelo DiffServ se encarrega de oferecer QoS provendo à rede classes de serviço e realizando a diferenciação de pacotes para *agregações de fluxos*.

A arquitetura DiffServ é descrita na RFC 2475 [5]. Nesse documento são especificadas a terminologia utilizada pela arquitetura, as características do modelo, os componentes e as classes de serviço que podem ser oferecidas na rede DiffServ. Nas próximas seções dedicaremo-nos a descrever este modelo.

2.1.3.1 Arquitetura DiffServ

A diferenciação de serviços é realizada por meio da classificação e do condicionamento de tráfego executados nos roteadores de borda para os fluxos de pacotes que ingressam no domínio DiffServ. A diferenciação é executada levando-se em conta o PHB (*Per-Hop Behavior*) que cada pacote deve receber segundo o *DS codepoint* marcado no campo DS do pacote.

O PHB está associado a uma classe de serviço e diz como o pacote deve ser tratado pelo roteador, ou seja, se o pacote deve ser encaminhado com prioridade, rebaixado de prioridade ou descartado. O *DS codepoint* é o código que representa a classe de serviço a qual o pacote pertence.

O *DS codepoint* é codificado utilizando-se 6 bits do campo ToS, ou do campo CoS, do cabeçalho do pacote IP das versões do IPv4 e IPv6, respectivamente, renomeando-o para campo DS. Os dois bits restantes não são utilizados. A figura 2.1 exemplifica a localização do campo DS no cabeçalho do pacote IPv4.

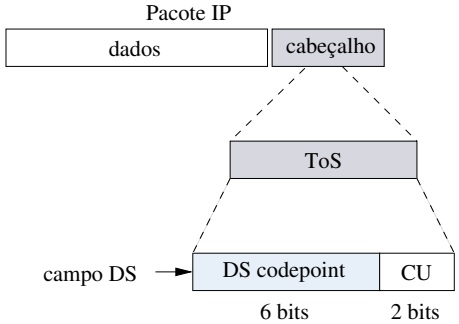

Figura 2.1: Campo DS DiffServ.

Um domínio DiffServ é composto por nós de borda (roteadores de ingresso e egresso) e pelos nós do núcleo da rede. Os nós de borda implementam as funcionalidades para classificação e condicionamento do tráfego que entra na rede. Os nós do núcleo se encarregam de encaminhar os pacotes de acordo com o PHB codificado no campo DS.

Do ponto de vista do gerenciamento da rede, deve existir um contrato de serviço entre a rede DiffServ e as redes clientes especificando como o tráfego da rede cliente deve ser tratado pelo domínio DiffServ. Esse contrato é chamado de SLA (*Services Level Agreement*). No SLA são especificadas as regras de classificação e condicionamento para o tráfego do cliente e, também, podem ser especificados os perfis de tráfego e as ações a serem tomadas para os fluxos que estão dentro ou fora do perfil. O contrato de condicionamento de tráfego TCA (*Traffic Conditioning Agreement*) é derivado do SLA, e especifica as regras para a medição, marcação, formatação e descarte dos pacotes que pertencem a uma classe de serviço.

Com o progresso do DiffServ, o SLA e o TCA representam um contrato de forma ampla incluindo os aspectos técnicos e de negócios (tarifação, contratos, etc). Portanto, uma nova terminologia deve ser utilizada para os contratos do ponto de vista técnico. O SLS (*Service Level Specification*) é um conjunto de parâmetros com seus valores, os quais, juntos, definem um serviço para um fluxo de tráfego. O TCS (*Traffic Conditioning Specification*) é um conjunto de parâmetros com seus valores, os quais, juntos, especificam um conjunto de regras de classificação e um perfil de tráfego. Um TCS é parte de um SLS [16].

Na continuação, descreveremos as características dos mecanismos que devem ser implementados nos elementos de um domínio DiffServ. Logo a seguir, na Seção 2.1.3.2, são descritas as classes de serviço que podem ser suportadas em uma rede DiffServ.

Classificação do tráfego

A seleção do tráfego é realizada pelo classificador. A classificação consiste em identificar um subconjunto do tráfego que deve receber tratamento diferenciado sendo condicionado e/ou remarcado para uma ou mais agregações de comportamento (BAs). Uma BA (*Behavior Aggregate*) é uma agregação de fluxos que pertencem à mesma classe de serviço. Existem dois tipos de classificadores:

Classificador BA, executa a seleção de pacotes baseando-se somente no campo DS do pacote IP.

Classificador MF - *Multiple Field*, executa a seleção de pacotes levando em conta diversos

campos do cabeçalho do pacote. Os campos para realizar a classificação podem ser: endereço de origem, endereço de destino, campo DS, porta de origem, porta de destino, ID (identificador) do protocolo, entre outros campos.

O classificador deve ser configurado por meio de um procedimento de gerenciamento que leve em conta o contrato TCS.

Condicionamento de tráfego

O condicionamento de tráfego é realizado executando-se a medição, formatação (*shaping*), policiamento e/ou a re-marcação para assegurar que o tráfego que entra ao domínio DiffServ esteja conforme as regras especificadas no TCS. O condicionamento do tráfego é realizado medindo-se o fluxo classificado contra um *perfil de tráfego* para verificar se os pacotes estão dentro ou fora do perfil especificado para uma classe de serviço. Por exemplo, um perfil baseado em um medidor *token bucket* pode assemelhar-se com:

```
codepoint X, utiliza o medidor token-bucket r, b
```

O perfil descrito acima indica que todos os pacotes marcados com o *DS codepoint "X"* devem ser medidos contra um medidor *token bucket* com taxa *"r"* e tamanho de rajada *"b"*. Neste caso, os pacotes que estão fora do perfil são pacotes que chegam quando não se tem mais fichas disponíveis no balde.

Diferentes ações de condicionamento podem ser tomadas para pacotes dentro ou fora do perfil. Os pacotes dentro do perfil são permitidos entrar no domínio sem serem condicionados dependendo da classe à qual pertencem. Os pacotes fora do perfil podem ser: enfileirados até que estejam dentro do perfil (formatação), descartados (policiamento), marcados com um novo *DS codepoint* (remarcação) ou encaminhados se houver recursos disponíveis.

O condicionamento de tráfego pode ser provido pelos seguintes elementos:

Medidor, realiza a medição do fluxo de pacotes contra um perfil de tráfego especificado em um TCS. O medidor passa informações para as outras funções do condicionamento para disparar uma ação para cada pacote que está dentro ou fora do perfil.

Marcador, a função do marcador é marcar o campo DS de cada pacote com um *DS codepoint*. No caso de um pacote estar fora do perfil é realizada a remarcação.

***Shaper* (formatação),** este elemento condiciona todos ou parte dos pacotes de uma determinada classe que estão fora do perfil para que estes, depois de aplicada a formatação, estejam conforme o perfil de tráfego definido para a classe.

Elementos de descarte, estes elementos se encarregam de descartar todos ou parte dos pacotes que estão fora do perfil. Esse processo é chamado de policiamento do fluxo de pacotes.

A figura 2.2 mostra o classificador e os elementos do condicionamento de tráfego. Note que o condicionamento de tráfego não necessariamente deve conter os quatro elementos. Por exemplo, no caso de não existir um perfil de tráfego definido para uma determinada classe, os pacotes pertencentes à classe somente passarão pelo classificador e o marcador de pacotes.

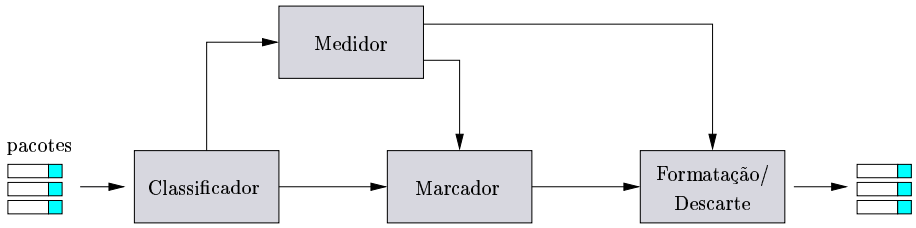

Figura 2.2: Classificação e condicionadomento de tráfego.

2.1.3.2 Classes de Serviço DiffServ

Atualmente existem duas classes de serviço que definem os PHBs que podem ser implementados nos roteadores do domínio DiffServ. Os PHBs definidos são: o PHB de encaminhamento expresso (*Expedited Forwarding*) e o PHB de encaminhamento assegurado (*Assured Forwarding*).

PHB de Encaminhamento Expedido - EF

O PHB EF é especificado na RFC 2598 [21]. Este PHB define uma classe de serviço com baixo atraso, baixa variação do atraso (*jitter*) e baixas perdas de pacotes, assegurando que as agregações EF sejam servidas com uma certa taxa r. Os pacotes que pertencem a esta classe de serviço são medidos contra um perfil de tráfego especificado com a taxa mínima r. Caso o pacote viole o perfil de tráfego, ele é automaticamente descartado.

PHB de Encaminhamento Assegurado - AF

O PHB AF é definido na RFC 2597 [17]. Este PHB define quatro sub-classes de serviço, cada uma delas com três níveis de precedência para descarte. Diferentemente da classe EF, os pacotes que pertencem a uma das sub-classes e violem o perfil de tráfego, são remarcados para uma sub-classe com menor prioridade. O descarte de pacotes se dá somente após a formatação do tráfego.

2.1.4 *Bandwidth Brokers*

Um *bandwidth broker* é uma entidade que basicamente gerencia os recursos de rede dentro de um domínio. O objetivo da utilização de um *bandwidth broker* numa rede, como por exemplo, uma rede DiffServ, é o de auxiliar no fornecimento de QoS da rede. O *bandwidth broker* também é responsável pelo gerenciamento da informação inter-domínios, realizado com os *bandwidth brokers* das redes vizinhas, com o propósito de coordenar os níveis de serviço (SLSs) nas fronteiras do domínio.

O *bandwidth broker* deduz e monitora os estados dos recursos de QoS dentro do seu domínio e sob as bordas dos domínios adjacentes. A informação captada no domínio, juntamente com a informação das políticas colhidas do repositório (base de dados das regras de políticas), é utilizada para tomar decisões de controle de admissão sobre as requisições de serviço para a rede. O gerenciador de políticas, se existe, verifica as requisições relativamente às regras de políticas instaladas, checando se há conflito com outras requisições e toma ações apropriadas de preempção. As informações do estado da rede são também utilizadas para verificar se há recursos disponíveis para suportar a requisição [27].

É importante ressaltar que o *bandwidth broker* e o gerenciador de políticas são duas entidades distintas, mas que se complementam na tomada de decisões sob requisições e podem estar implementados no mesmo sistema.

Características

Um *bandwidth broker* se encarrega de:

- gerenciar os recursos de rede de um domínio;
- controlar a admissão de um cliente;

- monitorar a rede e reagir de acordo com o estado em que esta se encontra;
- fazer com que os contratos e as políticas relacionados a uma classe de serviço sejam cumpridos com a ajuda do servidor de políticas;
- comunicar-se com *bandwidth brokers* de outros domínios adjacentes para realizar a reserva de recursos para uma aplicação que precise atravessar esses domínios, levando em conta o SLS estabelecido entre os domínios.

O *bandwidth broker* configura os roteadores de borda e do núcleo da rede com parâmetros associados a uma classe de serviço levando em conta os contratos estabelecidos com as redes clientes. Nestes contratos são definidas as políticas para o acesso aos recursos de rede.

A requisição de reserva de recursos de rede, realizada por uma solicitação ao *bandwidth broker*, inicia o processo de controle de admissão. O *broker* avalia os parâmetros solicitados no pedido, consultando a sua base de dados. Caso existam recursos disponíveis, ele aceita o pedido, senão, uma notificação é enviada para a aplicação informando que não há recursos disponíveis para garantir a transmissão. Neste caso, pode-se iniciar a transmissão dos dados da aplicação através do serviço de melhor esforço.

Arquitetura de um *Bandwidth Broker*

Segundo [27], a arquitetura de um *bandwidth broker* pode ser composta pelos seguintes elementos:

- Interface com o usuário/aplicação.
- Interface de comunicação inter-domínios.
- Interface de comunicação intra-domínios.
- Interface com a tabela de roteamento.
- Interface com o repositório de dados (políticas).
- Interface com o gerenciador de políticas.
- Interface com o gerenciamento de rede.

Não necessariamente todos os componentes são necessários para a implementação de um *bandwidth broker*. A figura 2.3 mostra os componentes da arquitetura de um *bandwidth broker*. No capítulo 3, é descrita a implementação do *bandwidth broker* utilizado pela plataforma MPLS−DS.

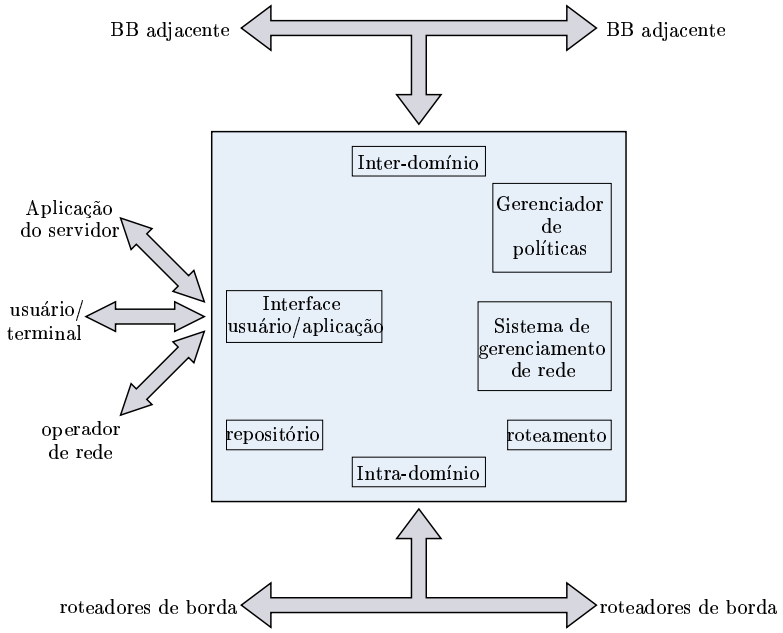

Figura 2.3: Arquitetura de um *Bandwidth Broker*.

Vantagens do *Bandwidth Broker*

A vantagem de se utilizar um *bandwidth broker* para o gerenciamento de recursos está na automatização da configuração dos recursos de rede nos roteadores do domínio. Atualmente, esta tarefa é feita por administradores de rede que configuram os roteadores manualmente. Os valores escolhidos para configuração são definidos através do histórico de utilização da rede e estas configurações podem demorar dias, semanas ou meses. O tráfego entre domínios vizinhos também é controlado desta forma, sendo que os valores são negociados através de meios tais como correio eletrônico, telefone ou outro meio que implique em intervenção manual [8].

Implementações de um *Bandwidth Broker*

Atualmente existem várias implementações de *bandwidth brokers* utilizadas para gerenciar e executar as políticas para o acesso aos recursos de rede disponíveis. Parte do projeto Internet2

[20] conta com a implementação de um protótipo de um *bandwidth broker* para a configuração dos roteadores de um domínio DiffServ. No projeto Qbone [31], é realizada uma discussão sob os requisitos da arquitetura Qbone na Internet2, além disso são descritas as características e a arquitetura que um *bandwidth broker* pode ter. A implementação realizada em [8] conta com um *bandwidth broker* que configura a rede DiffServ simulada através da utilização do simulador *NS*. Em [3] é implementado um *bandwidth broker* que interage com um servidor de políticas (PDP) para o estabelecimento de políticas de configuração de LSPs em uma rede MPLS.

Um dos componentes utilizados pela plataforma MPLS−DS (produto desta dissertação) é um *bandwidth broker* que gerencia os recursos da rede MPLS−DS simulada. A implementação do *bandwidth broker* é separada da rede MPLS−DS implementada no simulador de redes *NS*. A comunicação entre o *bandwidth broker* e o simulador é realizada através da arquitetura CORBA. O *bandwidth broker* interage com a rede MPLS−DS simulada no *NS* e com um modelo de políticas de três camadas. O objetivo da plataforma é validar as políticas de configuração estabelecidas para uma determinada rede através de simulação, para que posteriormente essas políticas possam ser empregadas em uma rede real. A plataforma é descrita no capítulo 3 e a sua utilização é apresentada no capítulo 4.

2.2 Engenharia de Tráfego

A Engenharia de Tráfego (TE) é um mecanismo que pode ser oferecido em uma rede com o objetivo de aumentar o seu desempenho através do controle e otimização do roteamento. Provê a habilidade para mover fluxos de tráfego para caminhos físicos menos congestionados (além dos caminhos mais curtos selecionados pelo IGP[3]) com a meta de distribuir o tráfego uniformemente pela rede. A TE é uma ferramenta poderosa que pode ser utilizada para balancear a carga do tráfego em vários enlaces, roteadores e comutadores da rede, de modo que nenhum destes componentes seja sobre-utilizado ou pouco utilizado. Deve ser vista como uma ferramenta de ajuda para a infra-estrutura de roteamento provendo informação adicional para encaminhar o tráfego por meio de caminhos alternativos.

A engenharia de tráfego pode ser utilizada para oferecer uma utilização eficiente da largura de banda disponível e aumentar as características do desempenho da rede minimizando a perda de pacotes, os períodos prolongados de congestionamento e maximizando o *throughput*[4].

[3] IGP, *Interior Gateway Protocol*
[4] *throughput*, vazão total de pacotes.

Na Internet, a engenharia de tráfego pode ser realizada levando em conta ações preventivas e/ou reativas. As ações preventivas são tomadas para evitar o congestionamento. Por outro lado, as ações reativas são consideradas para que o sistema reaja a situações indesejadas que já aconteceram.

A melhora da QoS devido a engenharia de tráfego pode ser observada através de parâmetros tais como: atraso, variação do atraso ou perda de pacotes, como também pela percepção humana. A engenharia de tráfego empregada em uma rede pode ser avaliada através de métodos analíticos, simulações ou por métodos empíricos baseados em medições.

Para um melhor planejamento das redes, devem ser feitas análises para a determinação das rotas que os protocolos de roteamento selecionarão para atender as demandas de tráfego. Em tecnologias como MPLS, a criação de LSPs pode ser controlada levando-se em conta parâmetros tais como políticas de configuração. Os LSPs podem ser estabelecidos manualmente (via gerenciamento), automaticamente em tempo real (utilizando funções de roteamento baseado em restrições implementadas nos roteadores) e *offline* (realizado por entidades externas, como por exemplo: *bandwidth brokers*). Por outro lado, também pode-se considerar algoritmos para o balanceamento da carga do tráfego como por exemplo: o algoritmo MATE (*MLPS Adaptive Traffic Engineering*) [13] e os algoritmos propostos em [6], onde são realizadas variações nas métricas utilizadas pelo MATE para a execução do balanceamento. Em [23] é proposta uma plataforma que visa prover aos projetistas uma metodologia de implementação e testes de forma a facilitar o trabalho de criação de novos algoritmos relacionados a engenharia de tráfego.

Na próxima Seção 2.2.1, descreveremos a arquitetura da rede MPLS e discutiremos porque esta tecnologia está sendo visada para prover a engenharia de tráfego para a rede Internet. Para finalizar este capítulo, na Seção 2.2.2, descrevemos como a diferenciação de serviços nas redes MPLS pode ser provida.

2.2.1 Redes MPLS - *MultiProtocol Label Switching*

A arquitetura da rede MPLS é especificada na RFC 3031 [36]. Basicamente, a rede MPLS é responsável por dirigir um fluxo de pacotes por um caminho pré-determinado dentro da rede. Este caminho é chamado de LSP (*Label Switched Path*). Os LSPs são similares aos PVCs ATM por sua simplicidade, pois os fluxos de tráfego seguem uma só direção desde o roteador de ingresso até o roteador de egresso. Um LSP é a concatenação de um ou mais *hops* que permitem ao pacote ser encaminhado desde um roteador comutador de rótulos no ingresso da rede até outro LSR de egresso (*Label Switching Router*), através do domínio MPLS. Os LSRs são roteadores que suportam

o encaminhamento MPLS, ou seja, a comutação baseada no rótulo do pacote.

Quando um LSR de ingresso recebe um pacote IP, ele adiciona ao pacote o cabeçalho MPLS (*shim header MPLS*) e encaminha o pacote para o próximo LSR no LSP. O pacote rotulado é encaminho no LSP através de cada LSR até ele chegar ao nó de egresso do LSP, neste ponto o cabeçalho MPLS é removido e o pacote é encaminhado baseado no endereço IP de destino. O ponto chave deste esquema é que o caminho físico seguido pelo LSP não é limitado pelo caminho escolhido pelo IGP, que é o caminho mais curto para cada endereço IP de destino.

O cabeçalho MPLS é composto por quatro campos <*rótulo, campo EXP, campo S, TTL*[5]> codificados em 4 bytes : o rótulo é formado por 20 bits, o campo EXP é composto por 3 bits que é referenciado como o campo experimental, o campo S é codificado em um bit, indicando se o rótulo atual é a ultima entrada na pilha de rótulos, e o campo TTL é formado por 8 bits cujo significado varia de acordo a seu conteúdo. Detalhes do significado de cada campo são obtidos em [35].

O processo de encaminhamento de pacotes é baseado na troca de rótulos (*label swapping*) realizado em cada LSR. Quando um pacote chega a um LSR já rotulado, o LSR examina o rótulo e o utiliza como índice da sua tabela de encaminhamento MPLS. Cada entrada na tabela de encaminhamento contém um par <*interface*, *rótulo de entrada*> que é mapeado para um conjunto de funções de encaminhamento que são aplicadas para todos os pacotes que chegam à *interface* com o mesmo *rótulo de entrada*.

Os protocolos que podem ser utilizados para efetuar a distribuição de rótulos a cada LSR em uma rede MPLS são o LDP (*Label Protocol Distribution*) [2] e o RSVP (*ReSource reserVation Protocol*) [4], este último sofre extensões para poder realizar a distribuição dos rótulos.

Benefícios do MPLS

O principal benefício que a tecnologia MPLS provê é a separação entre o roteamento (isto é, controle) e o encaminhamento (isto é, transporte de dados) [36]. Esta separação permite o emprego de algoritmos de encaminhamento e balanceamento de tráfego MLPS para poder realizar a engenharia de tráfego, que podem ser utilizados por múltiplos serviços e tipos de tráfegos. Além disso, pode-se criar LSPs que reúnam requisitos específicos de QoS, um grupo *multicast* IP, ou uma rede virtual privada (VPN). Desta maneira, novos serviços podem ser oferecidos e migrados para operar sob uma infra-estrutura de encaminhamento MPLS comum.

[5]TTL, campo *Time To Live*.

2.2.2 Redes MPLS com suporte à Diferenciação de serviços

A rede MPLS não provê a diferenciação de classes de serviço para aplicações que precisem que os seus pacotes recebam tratamento diferenciado dentro desta rede. Em [15] é proposto um mecanismo para que a diferenciação dos tráfegos associados às classes de serviço seja realizada na rede MPLS. Basicamente, a solução dada nesse documento consiste em construir dois tipos de LSPs (E−LSP e L−LSP), que inferem o comportamento (PHB[6]) a ser dado para um determinado pacote a partir do campo EXP ou do rótulo do pacote.

Os **E−LSPs** podem transportar várias classes de serviço. A classe de serviço de um pacote é inferida decodificando o conteúdo do campo EXP (campo *EXP*erimental) do cabeçalho MPLS. O campo EXP é formado por três bits, portanto podem ser transportadas até oito classes de serviço em um E−LSP. Para este tipo de LSP, a precedência de descarte para o pacote também é inferida utilizando-se o campo EXP.

Os **L−LSPs** somente podem transportar uma única classe de serviço. O comportamento PHB que se dá para um pacote é inferido decodificando parte do rótulo associado ao pacote e a precedência de descarte para o pacote é inferida no campo EXP do cabeçalho MPLS.

Também, deve ser oferecido um mecanismo para o mapeamento de classes de serviço nos LSPs criados. Ou seja, deve existir uma estrutura de dados contendo a informação relacionada às classes de serviço oferecidas na rede DiffServ *upstream* e as classes equivalentes oferecidas no domínio MPLS, denominado mapeamento PHB $\leftrightarrow$ EXP.

No Capítulo 3 é descrita a plataforma MPLS−DS. Parte da plataforma é formada por uma rede MPLS que provê a diferenciação de serviços como especificado acima (rede MPLS−DS), tornando possível a criação de E−LSPs e L−LSPs para o transporte dos fluxos de dados associados às classes de serviço. Desta maneira, realiza-se a diferenciação de serviços, por meio do DiffServ, e a engenharia de tráfego através do MPLS, contribuindo para que as características de uma determinada aplicação com QoS sejam cumpridas. Mais informações sobre a qualidade de serviço dentro de uma rede MPLS são encontradas em [43].

[6]PHB, comportamento por *hop*.

Capítulo 3

Plataforma MPLS−DS

Neste capítulo apresentaremos a plataforma MPLS−DS, proposta desta dissertação. Inicialmente, na Seção 3.1, justificaremos porque estamos realizando este trabalho e descreveremos o objetivo da plataforma. O simulador utilizado para a construção da plataforma é o *Network Simulator−NS* [14]. Na Seção 3.2, discutiremos as principais características do simulador, onde serão apresentados os seus componentes mais importantes. Na Seção 3.3, são descritas as características da plataforma e a sua funcionalidade. Depois especificaremos, nas Seções 3.4 e 3.5, os componentes da plataforma e a maneira como foram implementados, respectivamente.

3.1 Justificativa e objetivo

Justificativa

O interesse de se ter a rede Internet como uma rede multiserviços (como por exemplo: poder oferecer serviços para o comércio eletrônico, aplicações de voz, vídeo conferências, etc) deu motivo para que grupos de trabalho fossem criados no âmbito da IETF, dando origem a novas tecnologias capazes de realizar a diferenciação de serviços e, paralelamente, aumentar o desempenho da rede. Essas novas características originaram dois termos bastante utilizados atualmente no contexto da rede Internet: *Qualidade de Serviço* e *Engenharia de Tráfego*, vistos no capítulo anterior.

Neste sentido, os Serviços Diferenciados−DiffServ e as redes MPLS são tecnologias que podem prover a diferenciação de serviços e a engenharia de tráfego para a rede Internet de uma maneira escalável, como foram descritas nas Seções 2.1.3 e 2.2.1, respectivamente.

Isto nos levou a propor uma plataforma que simule a rede Internet e que forneça a tecnologia

MPLS acoplada à capacidade de diferenciação de serviços. Conseqüentemente, será necessário que exista um mecanismo que configure a rede e os parâmetros de qualidade de serviço. O esquema de configuração agirá de acordo com as políticas de configuração e os contratos assinados com clientes especificando o serviço que será oferecido. Portanto, nossa proposta consiste em construir uma plataforma para a rede Internet que possa:

- fornecer um mecanismo para configurar os roteadores do *backbone* baseando-se em políticas de configuração de rede e levando em conta o contrato SLS[1] estabelecido com uma rede cliente para a aplicação da política,
- prover ao *backbone* da rede a diferenciação de tráfegos de distintas aplicações (prover mecanismos para distinguir as classes de serviço) e fazer com que a rede seja melhor utilizada (aumentar o seu desempenho).

Objetivo

O objetivo principal da plataforma MPLS−DS é permitir a validação de políticas de configuração de uma rede por meio de simulações, para que essas políticas possam ser empregadas posteriormente no *backbone* de uma rede real.

Para poder realizar isso, a rede deve ser formada por elementos que possam: diferenciar o tráfego, criar rotas explícitas no *backbone*, configurar e gerenciar a rede através de políticas de configuração. A plataforma MPLS−DS provê todos esses elementos à rede, como veremos nas próximas seções.

Em segundo lugar, o *bandwidth broker* construído como um mecanismo de gerenciamento da rede simulada, independente do simulador *NS*, poderá ser visto como uma ferramenta de gerência que possa ser empregada em uma rede real para executar o gerenciamento dos recursos de rede de forma automatizada.

3.2 O Simulador *NS*

O simulador de redes *NS* [28] (*Network Simulator*) surgiu do projeto VINT [41] (*Virtual InterNetwork Testbed*). O objetivo deste projeto foi criar um simulador para a pesquisa de redes que permita estudar a interação e escalabilidade dos protocolos de redes atuais e futuros.

[1] SLS, *Service Level Specification.*

Este projeto é financiado pela DARPA [9] (*Defense Advanced Research Projects Agency*) e conta com a colaboração do instituto *USC/ISI* [40] do centro de pesquisa *Xerox PARC* [44], do laboratório *LBNL* [22] e da universidade *UC Berkeley* [39]. O *NS* é um simulador de domínio público.

Na próxima seção apresentaremos as principais características do simulador. Na Seção 3.2.2, descreveremos a arquitetura do simulador e na Seção 3.2.3 são expostos os principais componentes do simulador e das ferramentas auxiliares ao *NS*.

3.2.1 Características do *NS*

O *NS* é um simulador de redes de comunicação baseado em eventos discretos, implementado em duas linguagens C++ e OTcl (*Object Tool Command Language*). Este simulador tem destaque na pesquisa das redes de comunicações e provê suporte para simulação de protocolos de roteamento, protocolos TCP e UDP, e protocolos de *multicast*, tanto em redes convencionais como em redes sem fio. Abaixo listamos as características e as vantagens oferecidas por este simulador.

Sistema aberto, o *NS* é um sistema com código fonte aberto, que nos permite estender as classes existentes, ou mesmo criar novas classes, segundo a necessidade da rede a ser simulada, tornando-se flexível para a criação/modificação dos protocolos e da funcionalidade da rede.

Simulador com programação orientada a objetos, o *NS* é baseado nas características da programação orientada a objetos, o que facilita a criação de novos componentes (novas classes) de rede, como por exemplo: protocolos, classificadores, filas, etc.

O *NS* é compilado e interpretado, os protocolos de roteamento e as funcionalidades que requerem de alto processamento de dados são compilados. Por outro lado, a configuração da rede a ser simulada é interpretada.

Robustez, o *NS* é um programa robusto que provê suporte para uma variedade de protocolos e é bastante utilizado no meio acadêmico como uma ferramenta para a pesquisa das redes de comunicações.

As simulações detalhadas dos protocolos requerem sistemas de linguagens de programação que possam manipular eficientemente *bytes*, cabeçalhos dos pacotes, e implementar algoritmos que processam grandes conjuntos de dados. Para essas tarefas, a velocidade no tempo de execução é importante. Por outro lado, a configuração da rede (topologia, aplicações que geram o tráfego da

rede, configuração das filas, etc) pode mudar os seus parâmetros de configuração para a exploração rápida de cenários. Nestes casos, o tempo de iteração (mudar o modelo e executá-lo novamente) é mais importante.

O *NS* atende cada um destes requisitos através do uso das linguagens, C++ e OTcl. A linguagem C++ é rápida para executar mas lenta para mudar e analisar o código implementado, favorecendo as implementações detalhadas dos protocolos. Um programa escrito em OTcl executa mais lentamente pelo fato de tratar-se de uma linguagem interpretada mas, por outro lado, pode ser alterado rapidamente. A linguagem OTcl é utilizada para criar o arquivo de configuração da rede.

3.2.2 Arquitetura do *NS*

O simulador está estruturado por uma hierarquia de classes criadas em C++ (chamada de hierarquia compilada) e uma hierarquia de classes similar dentro do interpretador OTcl (chamada hierarquia interpretada). Os usuários criam novos objetos de simulação, por meio de *scripts*, a serem simulados através do interpretador. Estes objetos são instanciados dentro do interpretador e são espelhados por objetos correspondentes na hierarquia compilada. Via *tclcl*, o *NS* provê a junção para fazer com que os objetos e as variáveis criadas apareçam em ambas hierarquias (compilada e interpretada).

A arquitetura do simulador *NS* está dividida em quatro camadas (figura 3.1): camada de usuário, camada OTcl, camada de ligação, e a camada C++.

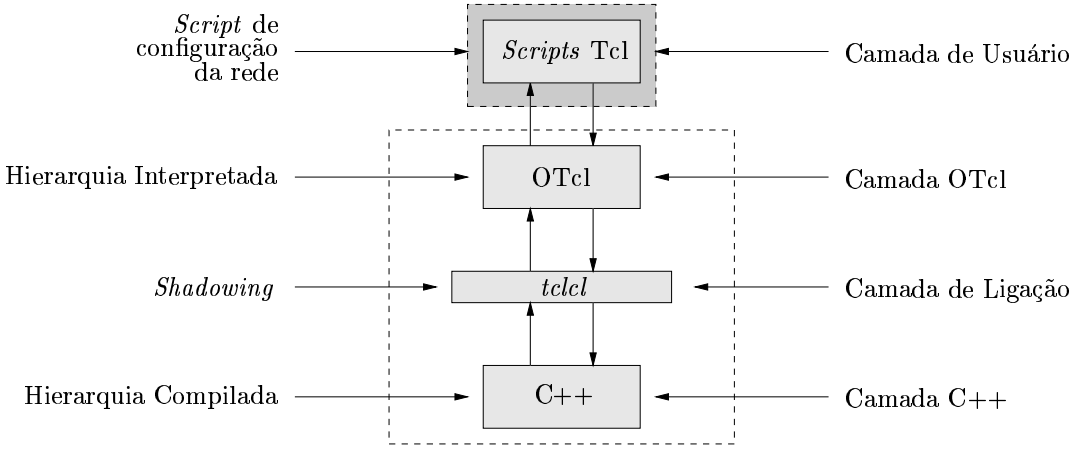

Figura 3.1: Camadas da arquitetura *NS*.

Camada de Usuário.- Nesta camada, o usuário realiza as configurações da rede que deseja simular por meio de um *script*, utilizando as APIs[2] existentes para configurar o núcleo da rede (camada C++). Neste nível se especifica o número de nós que possui a rede, a capacidade dos enlaces, o número de agentes geradores de tráfego ligados a um nó, entre outras informações.

Camada OTcl.- As APIs especificadas no *script* de configuração da rede pelo usuário são interpretadas por esta camada. Nela são criadas as instâncias para os objetos da rede (nós, enlaces, filas, etc.).

Camada *tclcl*.- Esta camada está encarregada de efetuar as ligações entre os objetos OTcl, instanciados pelos *scripts* dos usuários, e os objetos C++ espelhados na camada C++. Nesta camada também são espelhadas as variáveis instanciadas nos *scripts* de configuração da rede.

Camada C++.- Esta camada também é chamada de núcleo. O alto processamento de dados, algoritmos de roteamento, a classificação de pacotes, entre outras operações realizadas numa rede, são executadas nesta camada.

3.2.3 Componentes do *NS*

O simulador *NS* possui cinco componentes básicos para poder realizar uma simulação de rede. Estes componentes são o nó *NS*, que simula as características de um roteador, os enlaces, as filas, os geradores de tráfego e os monitores para coleta de dados. No *script* abaixo exemplificamos como são criados estes componentes através das APIs implementadas em OTcl.

```
set ingresso  [$ns node]
set egresso   [$ns node]
set largura_de_banda 2.88Mb
set atraso_do_enlace 2ms
$ns duplex-link $ingresso $egresso $largura_de_banda $atraso_do_enlace DropTail
```

Na figura 3.2, visualizamos os nós de ingresso, egresso e o enlace duplo referenciados no *script* acima. Na criação de um nó implicitamente se criam as filas, classificadores e escalonadores de pacotes. Como descrito no *script*, a fila utilizada para a simulação do exemplo é uma fila do tipo *DropTail*[3]

[2] API, *Application Programming Interface.*
[3] *DropTail*, fila do tipo FIFO.

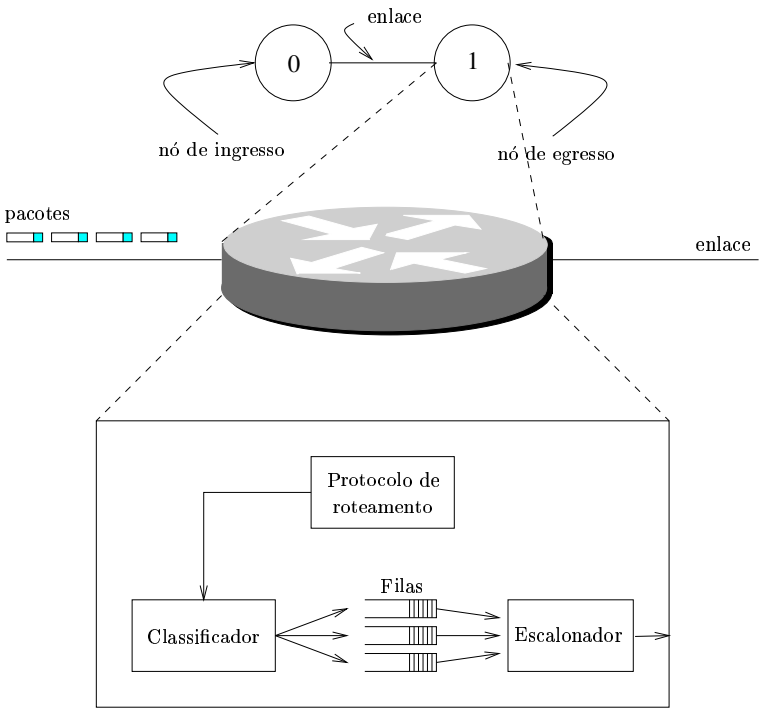

Figura 3.2: Componentes do *NS*.

É importante ressaltar que os enlaces possuem agentes de monitoramento que são utilizados para coletar dados relacionados ao desempenho da rede, como por exemplo, atraso e descarte de pacotes, total de utilização da banda, entre outras medições.

Existem duas ferramentas auxiliares ao *NS* que permitem visualizar o que está acontecendo na rede simulada. A primeira se refere ao animador de rede "*nam*" (*network animator*), que é utilizado para visualizar a topologia da rede simulada, a transferência de pacotes, visualização dos enlaces com falha, pacotes descartados, entre outros recursos. A segunda ferramenta, é o programa "*xgraph*" que permite visualizar arquivos de monitoramento que contém dados do desempenho da rede, como por exemplo: atraso e perda de pacotes, utilização de banda, porcentagem de pacotes transmitidos, etc.

3.3 Descrição da plataforma MPLS−DS

Contexto da plataforma

A plataforma MPLS−DS foi projetada para ser utilizada no *backbone* da rede Internet para prover a diferenciação de tráfegos de aplicações distintas, a criação de LSPs associados às classes de serviço e um mecanismo de configuração e gerenciamento baseado em políticas de configuração e um contrato SLS.

A plataforma conecta os domínios vizinhos (DiffServ e não-DiffServ), realizando o mapeamento das classes de serviço oferecidas nos domínios DiffServ para classes de serviço existentes na rede MPLS−DS[4] da plataforma. Além disso, na plataforma se cumpre o contrato SLS estabelecido com a rede DiffServ cliente que especifica como será realizada a classificação, o tratamento e policiamento de fluxos de pacotes marcados. O mapeamento de classes de serviço, a criação de LSPs e a configuração das filas são realizados baseando-se em políticas de configuração de rede. A figura 3.3 mostra este esquema.

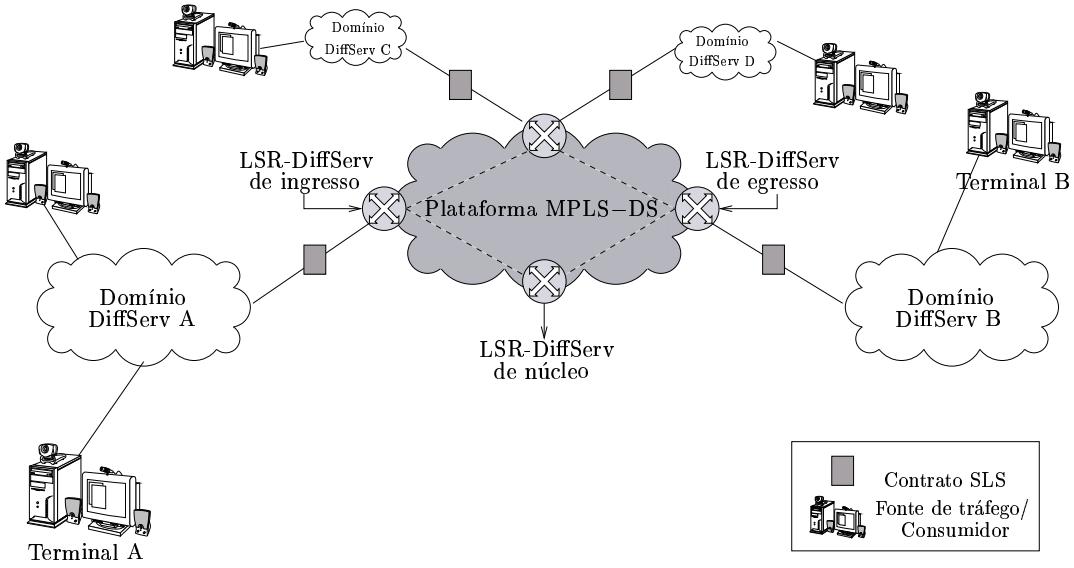

Figura 3.3: Contexto de atuação da plataforma MPLS−DS.

A seguir descreveremos as características da plataforma MPLS−DS e apresentaremos a sua funcionalidade.

[4]rede MPLS−DS, rede MPLS com suporte a diferenciação de serviços.

Características da plataforma

As principais características da plataforma são:

- Os mecanismos de configuração e gerenciamento de recursos de rede são executados por um *bandwidth broker* baseado em políticas de configuração e no contrato de serviços firmado com uma rede cliente.
- A validação das políticas de configuração, empregadas para uma rede, é realizada através de simulações, utilizando uma arquitetura de políticas e um *bandwidth broker* que interage com a rede dinamicamente.
- A plataforma MPLS−DS realiza o mapeamento das classes de serviço oferecidas em domínios DiffServ vizinhos para uma classe de serviço equivalente dentro da plataforma, denominado mapeamento PHB $\leftrightarrow$ EXP.
- Conforme já mencionado anteriormente a implementação da plataforma foi realizada com a utilização do simulador *NS*. Esta característica é descrita na Seção 3.5.

Como indicado na Seção 2.2.2, é necessário que exista um mapeamento para que os fluxos de tráfego que chegam marcados ao domínio da plataforma MPLS−DS, isto é, fluxos que pertencem a uma classe de serviço, continuem sendo tratados de maneira similar/equivalente a como eram tratados no domínio *upstream*, tornando necessário que a plataforma forneça um mecanismo que se encarregue do mapeamento de serviços. Na figura 3.4, os pacotes da classe x que deixam a rede DiffServ são mapeados para a classe y dentro da plataforma MPLS−DS.

O *bandwidth broker* se encarrega de passar informações de configuração para os roteadores do *backbone* e também recebe requisições dos PEPs[5] para que aloque recursos de rede para uma determinada classe. No roteador, existe um elemento que manda informação para o *bandwidth broker* e recebe informação deste para poder configurar o roteador.

Implementações das arquiteturas MPLS e DiffServ são disponíveis no simulador *NS* e são utilizadas pela plataforma, como veremos na Seção 3.5.

[5]PEP, elemento da arquitetura de políticas.

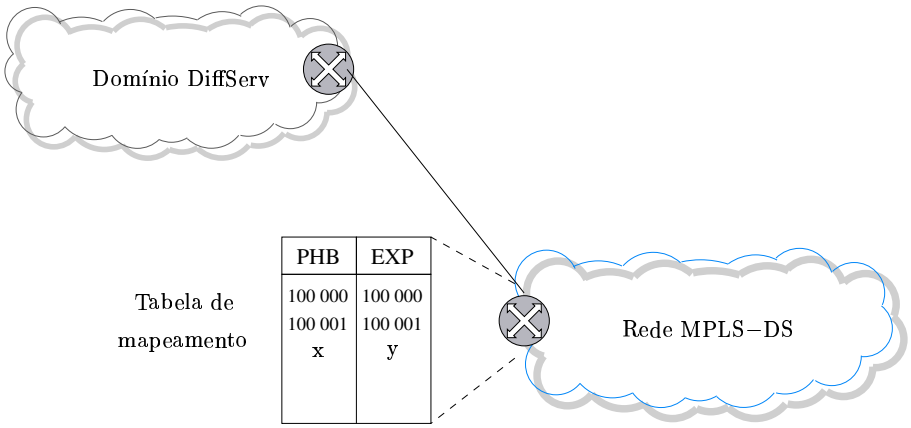

Figura 3.4: Mapeamento das classes de serviço.

Funcionalidade da plataforma

A plataforma MPLS−DS é uma ferramenta que auxilia ao administrador da rede na configuração e gerenciamento dos recursos de rede, isto é: filas, escalonadores, etc. Também, ajuda a configurar e gerenciar os LSPs e as classes de serviço existentes na plataforma. As principais funcionalidades da plataforma são:

- automatização da configuração dos recursos de rede de forma dinâmica através da utilização do *bandwidth broker*, e
- testar o desempenho da rede com a utilização das tecnologias propostas, o que permite realizar uma análise rápida do comportamento da rede, contribuindo para o melhor desempenho desta.

3.4 Componentes da plataforma MPLS−DS

A plataforma MPLS−DS é composta por cinco elementos: um *Bandwidth Broker*, a rede MPLS−DS, o mecanismo de mapeamento das classes de serviço, a arquitetura de políticas e pelos contratos SLSs. Estes elementos estão distribuídos nos planos de gerenciamento, controle e dados. Do ponto de vista do plano de gerenciamento temos o *bandwidth broker* e o PDP[6]. O plano de controle é formado por uma rede MPLS que provê a diferenciação de serviços (rede MPLS−DS),

[6]PDP, elemento da arquitetura de políticas.

um mecanismo de mapeamento de classes de serviço e pelo PEP. O plano de dados está formado pelo repositório de políticas e os contratos SLSs estabelecidos com as redes clientes. A figura 3.5 mostra os componentes da plataforma.

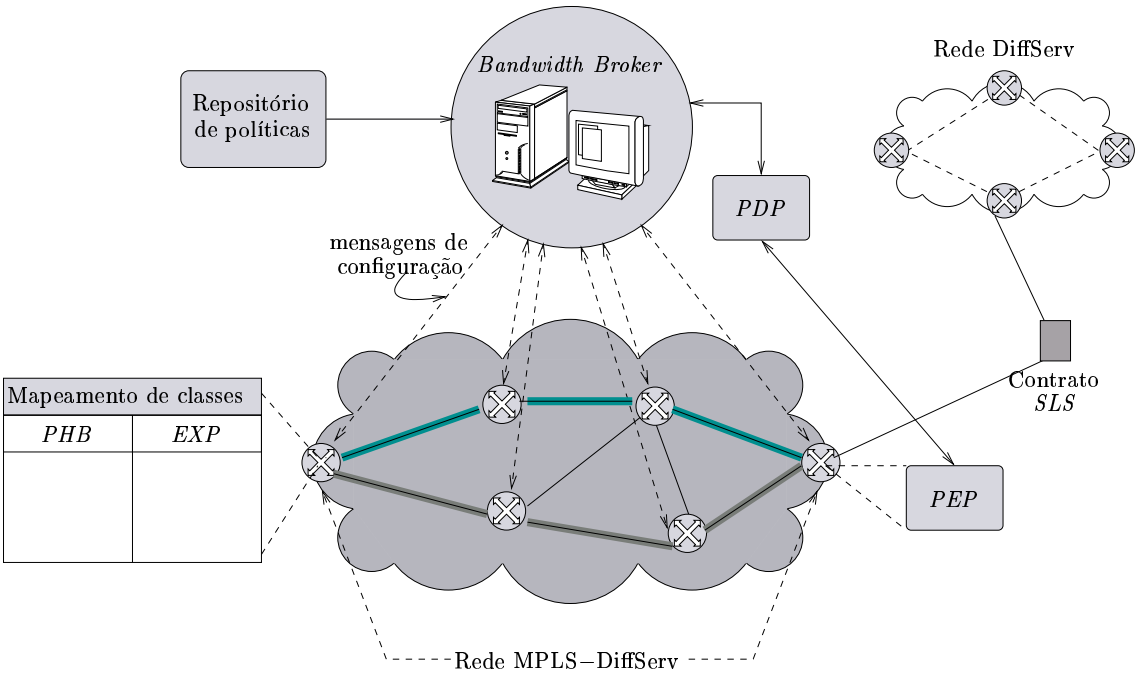

Figura 3.5: Componentes da plataforma MPLS−DS.

A seguir descreveremos as características de cada um dos componentes.

Bandwidth Broker.- O *bandwidth broker* é encarregado de gerenciar os recursos de rede, controlar a admissão dos clientes, monitorar a rede dinamicamente e de reagir de acordo ao estado em que esta se encontra. Além disso, faz com que as políticas relacionadas a uma classe de serviço sejam cumpridas. O *bandwidth broker* e o PDP são dois elementos que se complementam e dependendo da implementação realizada, podem estar localizados no mesmo elemento de rede.

Arquitetura de políticas.- Uma arquitetura de políticas é um conjunto de modelos de informação, protocolos e serviços, que permitem traduzir as intenções administrativas de como realizar a diferenciação dos fluxos de pacotes que atravessam uma rede. O modelo de informação da arquitetura de políticas utilizado na plataforma está baseado no modelo de política de três camadas (*Three− Tier Policy Model*) [34]. Os elementos deste modelo são o PDP (*Policy Decision Point*), o PEP (*Policy Enforcement Point*) e o *repositório de políticas*, conforme

mostrado na figura 3.6. A seguir citamos as principais características dos componentes desse modelo.

PDP.- É o componente responsável pelas ações que serão aplicadas a cada pacote. O PDP interpreta as regras das políticas armazenadas no repositório baseando-se nas condições da rede (estado dos enlaces, se saturados ou não), como também nas informações recebidas dinamicamente (monitoramento dinâmico da rede). O PDP é o servidor de políticas para o *bandwidth broker*. O PEP, por meio do *bandwidth broker*, pode requisitar ao PDP para que este tome decisões no seu nome quando ocorrer algum evento específico, como por exemplo, na notificação de congestionamento o *bandwidth broker* tomará alguma decisão para contornar este fato consultado o PDP. Após a consulta ao PDP, as informações de reconfiguração são enviadas aos PEPs de cada roteador por meio do *bandwidth broker*.

PEP.- O PEP é o elemento que realmente lida com os pacotes e é o responsável pelo cumprimento e execução das ações das políticas. O PEP é um componente operacional que pode tomar ações como: classificação e marcação de pacotes, cumprimento das taxas de tráfego, formatação do tráfego (*shaping*), gerenciamento de recursos, entre outras atividades. O PEP também pode solicitar ao PDP para que tome decisões no seu nome o qual, após decidir sobre a solicitação, manda as informações para reconfigurar o roteador. Os PEPs estão localizados nos nós de borda como nos nós do núcleo da rede.

Repositório de políticas.- O repositório de políticas armazena as políticas de configuração para o domínio. Pode estar localizado em um único lugar físico dentro do domínio onde as políticas fazem sentido ou pode estar duplicado em vários dispositivos. O repositório pode ser uma base de dados, um arquivo único, um servidor administrativo, ou um servidor de diretórios. As políticas são armazenadas no repositório por meio de uma ferramenta de gerenciamento de políticas ou manualmente. A ferramenta de gerenciamento de políticas também pode prover funções para validar as políticas armazenadas no repositório. Na nossa plataforma, o repositório de políticas é um arquivo e as políticas são armazenadas manualmente.

As três camadas descritas acima devem trocar informações. A comunicação entre o PDP e o PEP pode ser realizada através de vários protocolos: COPS (*Common Open Policy Service*), CLI (*Command Line Interpreter*) ou SNMP (*Simple Network Management Protocol*). Da mesma forma, a comunicação entre o PDP e o repositório de políticas pode ser realizada por vários protocolos, dependendo da natureza do repositório de políticas. Quando o repositório

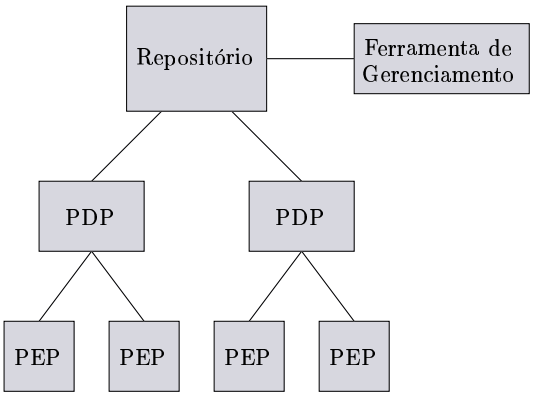

Figura 3.6: Modelo de política *Three− Tier*.

de políticas é um diretório de rede, o protocolo utilizado é o LDAP (*Lightweight Directory Access Protocol*).

O protocolo COPS é um protocolo padrão para troca de informações de políticas e decisões utilizado entre o PDP e PEP. Este protocolo foi projetado para trabalhar confiavelmente e em tempo real, com uma sobrecarga mínima. A comunicação entre o PDP e o repositório pode ser realizada através do protocolo LDAP.

Na nossa implementação, o PDP e o PEP se comunicam através de chamadas de métodos. As requisições ao repositório são realizadas acessando diretamente o arquivo que contém as informações sobre as políticas.

Rede MPLS com DiffServ.- O *backbone* da rede Internet é composto por duas tecnologias: MPLS e DiffServ, que chamaremos de rede MPLS−DS, possibilitando a diferenciação de classes de serviço e a criação de dois tipos de LSPs, E−LSPs e L−LSPs, como descrito na Seção 2.2.2.

Mapeamento das Classes de Serviço.- O mecanismo de mapeamento das classes de serviço, como o próprio nome diz, se encarrega de associar as classes de serviço oferecidas em uma rede DiffServ para uma classe equivalente oferecida na rede MPLS−DS. A decisão do mapeamento das classes de serviço é tomada de acordo com as políticas armazenadas no repositório e com o contrato assinado entre a rede cliente e a plataforma.

Contrato SLS.- O SLS é a especificação de níveis de serviços que contém informação relacionada ao *bandwidth broker* e aos dispositivos de rede para suportar um SLA[7] na rede. Um SLA é um

[7]SLA, *Service Level Aggrement*.

contrato de serviços firmado entre uma rede cliente e a provedora de serviços que especifica o serviço de encaminhamento que um cliente deverá receber. A informação contida em um SLS é aplicada às agregações de fluxos de dados e aos recursos de rede ou banda providos para esses fluxos. Um SLS é tipicamente aplicado nos nós de borda da rede. O SLS é a tradução do SLA em um nível compreensível para os dispositivos da rede. Parte do SLA é o contrato de condicionamento de tráfego (TCA) no qual se especifica como o tráfego do cliente deverá ser condicionado quando entra na rede MPLS−DS.

A seguir, detalharemos a maneira como foi implementada a plataforma MPLS−DS. Também, explicaremos as funções das APIs (*Application Programming Interfaces*) inseridas para a utilização da plataforma no simulador *NS*.

3.5 Implementação da Plataforma MPLS−DS

Na construção da plataforma, o primeiro passo dado foi o de integrar e condicionar as implementações das arquiteturas MPLS, denominada MNS [1], e DiffServ, denominada DS−Nortel [29], realizadas para o simulador *NS*, possibilitando a diferenciação de classes de serviço na rede MPLS. A segunda fase foi dedicada à construção do mecanismo para mapeamento de classes de serviço providas na plataforma. Na terceira fase da implementação construímos a arquitetura de políticas.

Inicialmente, apresentaremos brevemente na Seção 3.5.1, as implementações MNS e DS−Nortel que são utilizadas para construir a rede MPLS−DS. A implementação da rede MPLS−DS é descrita na Seção 3.5.2. O mecanismo de mapeamento das classes de serviço é explicado na Seção 3.5.3. A implementação da arquitetura de políticas é descrita na Seção 3.5.4. Na Seção 3.5.5, explicamos como são traduzidos os contratos e as regras de políticas a serem cumpridas pela plataforma.

3.5.1 Implementações MNS e DS−Nortel

Implementação MNS

O MNS (*MPLS Network Simulator*) é a implementação da arquitetura MPLS para o simulador de redes *NS*. O protocolo para distribuição de rótulos utilizado pelo MNS é o LDP (*Label Distribution Protocol*). Basicamente, o MNS é formado por dois elementos: por um *nó LSR*, que é um novo tipo de nó de rede dentro do simulador, e um agente LDP que simula o protocolo LDP.

As classes pertencentes ao módulo MNS são: *MPLSAddressClassifier*, *LDPAgent* e *MPLSModule*. Esta última classe diferencia o nó LSR de um nó convencional *NS* e liga ao nó um agente LDP para que ele possa compreender as mensagens LDP de sinalização. A figura 3.7, mostra as classes que pertencem ao módulo MNS com seus métodos mais importantes.

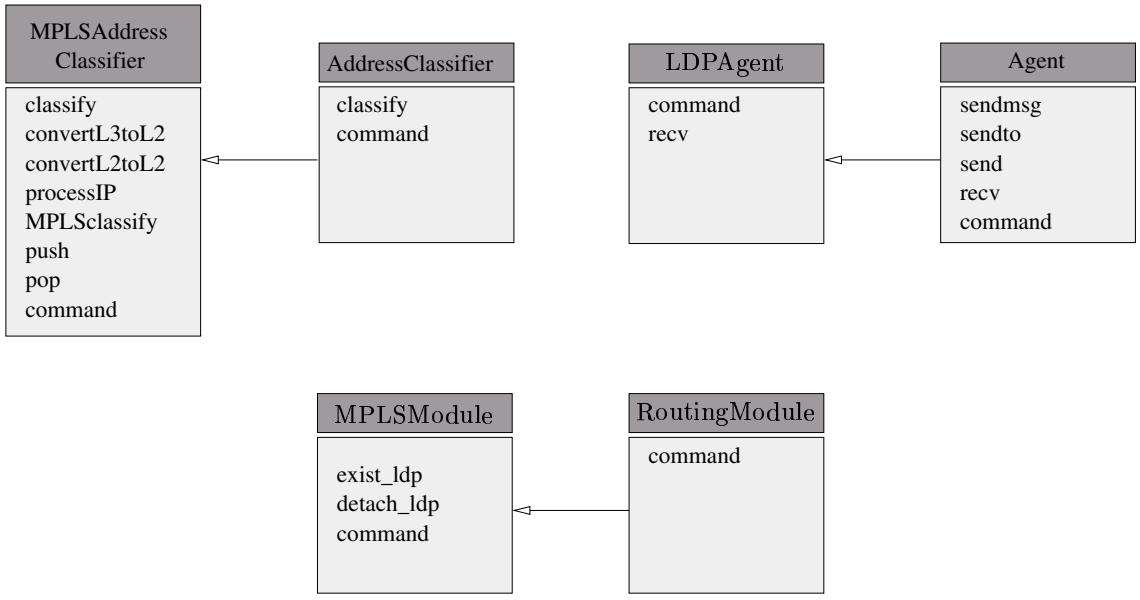

Figura 3.7: Classes do módulo MNS.

De maneira geral, o MNS oferece ao simulador *NS* a possibilidade de configurar nós LSRs que realizam a distribuição de rótulos por meio do protocolo LDP. Portanto, os LSRs comutam os pacotes baseando-se no rótulo contido no cabeçalho MPLS do pacote. Além disso, o MNS também oferece a criação de LSPs explicitamente em um domínio MPLS para um determinado tráfego. Todas essas funcionalidades de um nó LSR são encapsuladas no módulo *MPLSModule*.

A seguir, descreveremos brevemente as características do nó LSR e do agente LDP e, para finalizar esta seção, especificaremos as mudanças realizadas no MNS para poder fornecer a QoS nas redes MPLS simuladas. Detalhes da arquitetura MNS são apresentados em [1].

Nó LSR.- Este componente se encarrega de encaminhar o pacote baseado no rótulo inserido no cabeçalho MPLS (*shim header MPLS*). O processo seguido por este elemento é o de verificar se já existe um rótulo associado à FEC[8] do pacote. Caso positivo, é designado para o pacote o rótulo que corresponde à FEC, segundo a tabela de encaminhamento por rótulos

[8]FEC, *Forwarding Equivalence Class.*

ILM[9], e logo o pacote é encaminhado para o próximo nó. Caso contrário, o LSR atribuirá um rótulo para o pacote e logo o encaminhará para o próximo nó, baseando-se na tabela FTN[10]. Além de encaminhar o pacote rotulado ou não-rotulado para o próximo nó, o nó LSR mantém atualizadas as tabelas de rotas e de rótulos. Todas essas funcionalidades foram implementadas na classe *MPLSAddressClassifier*.

Agente LDP.- O agente LDP realiza a distribuição de rótulos entre os nós LSR da rede simulada. Esta implementação possui uma série de APIs que configuram a rede MPLS como desejado, como por exemplo, se a distribuição dos rótulos será feita sob demanda ou não, quem são os vizinhos LDP de um LSR baseado no protocolo LDP, entre outras. As APIs utilizadas normalmente pela plataforma MPLS−DS são a de criação de uma rota explícita e a de instalação de fluxos em um LSP. Todas essas funcionalidades foram implementadas na classe *LDPAgent*.

Modificações realizadas

A principal mudança realizada no código MNS foi passar a interpretar o campo EXP do cabeçalho MPLS. O interesse por interpretar este campo vem da utilização do mesmo para poder criar LSPs que consigam transportar até oito classes de serviço, como especificado na Seção 2.2.2. A interpretação consiste em diferenciar as classes de serviço através do código armazenado no campo EXP.

Implementação DS−Nortel

A implementação da arquitetura dos serviços diferenciados é chamada DS−Nortel. A arquitetura DS−Nortel foi construída por um grupo de pesquisa da empresa *Nortel Networks* [29]. A diferenciação de serviços se baseia na utilização de quatro filas físicas, sendo que cada uma delas possui três filas virtuais. Além disso, esta implementação conta com mecanismos para classificação e marcação de pacotes, para o policiamento, adequação e condicionamento do tráfego que entra na rede DiffServ. As classes do módulo DS−Nortel são: *redQueue*, *Policy*, *dsREDQueue*, *edgeQueue* e *coreQueue*, como mostrado na figura 3.8.

A classe *redQueue* é a classe mãe da classe *dsREDQueue*, a qual, por sua vez, é a classe mãe para as classes *edgeQueue* e *coreQueue*. A classe *redQueue* implementa as características das

[9]ILM, *Incoming Label Map*.
[10]FTN, *FEC to NHLFE Map*.

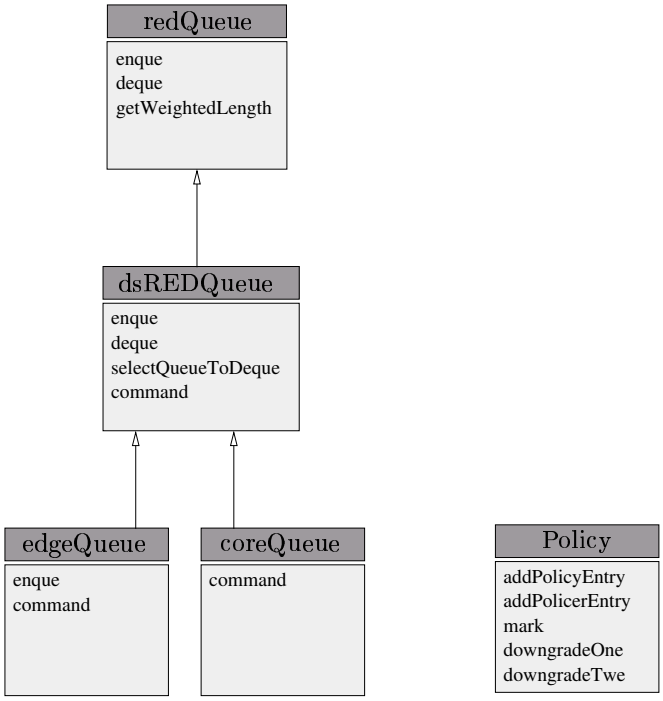

Figura 3.8: Classes do módulo DS−Nortel.

filas RED (*Random Early Detection*) e de algumas das variações deste tipo de fila (WRED, RIO, RIO-C, RIO-D e DROP). A classe *dsREDQueue* implementa e declara todas as funcionalidades e os parâmetros comuns para os roteadores de borda e de núcleo.

Os roteadores de borda realizam a classificação, marcação, condicionamento e policiamento do tráfego através da classe *edgeQueue*. Os roteadores do núcleo da rede se encarregam de encaminhar e tratar os pacotes de acordo com o PHB da classe à qual o pacote pertence. A classe *Policy* implementa as funções necessárias para realizar o policiamento e condicionamento dos fluxos de pacotes marcados.

Modificações realizadas

Um ponto negativo da implementação DS−Nortel é a necessidade de especificar obrigatoriamente os parâmetros referentes a QoS para os pacotes pertencentes à classe de melhor esforço. Portanto, a modificação realizada foi automatizar a configuração dos parâmetros de QoS para essa classe, deixando-a como classe padrão para todos os fluxos que entrem na rede sem receber nenhuma diferenciação. Ou seja, se um determinado fluxo não for associado a nenhuma classe de serviço

(classes EF ou AF) oferecida na rede, ele será associado à classe de melhor esforço.

3.5.2 Implementação da rede MPLS−DS

A implementação da rede MPLS−DS é formada pelas classes *MPLSDSAddressClassifier*, *MPLSDSModule*, *mplsREDQueue*, *mplsedgeQueue*, *mplscoreQueue* e *mPolicy*, que herdam as características das classes dos módulos MNS e DS−Nortel (ver figura 3.9).

As novas classes criadas herdam as características das redes MPLS e DiffServ. Essas características refere-se às habilidades de poder criar LSPs e configurar as filas e as classes de serviço dentro da rede MPLS−DS.

Com esta implementação, passa-se a interpretar o campo EXP do cabeçalho MPLS para poder dar tratamento diferenciado aos pacotes que chegam ou são marcados na borda com algum código relacionado a uma classe de serviço.

Também, pode-se criar dois tipos de caminhos comutados por rótulos, E−LSPs e L−LSPs. A referência [15] propõe que um E−LSP pode possuir até oito classes de serviços codificados no campo EXP, e um L−LSP somente pode transportar uma classe de serviço codificada no rótulo do pacote.

Com a combinação de ambas implementações passamos a ter roteadores LSR-DiffServ no simulador capazes de diferenciar os pacotes pertencentes a classes de serviço que são encaminhados através de um LSP.

Também foi necessário criar uma classe para representar o módulo MPLS−DS, implementado na classe *MPLSDSModule*. Este módulo possui as APIs necessárias para poder configurar os roteadores LSR−DiffServ da rede. Esse módulo é chamado de *MPLS/DiffServ* e é utilizado com a API *get-module*, como podemos ver no código abaixo.

```
$ns at $tempo "[$LSR1 get-module "MPLS/DiffServ"] ... "
```

O objetivo desta implementação é ter uma entidade que atue no plano de gerenciamento da rede (um *bandwidth broker*) e que se encarregue de configurar e gerenciar os recursos de rede. As APIs dos módulos MNS e DS−Nortel são herdados por este módulo.

De forma geral, a rede MPLS−DS utiliza as APIs definidas nas implementações MNS e DS−Nortel herdadas pelo módulo *MPLSDSModule* como também as que são definidas neste módulo. A seguir, listaremos as APIs mais importantes utilizadas pela plataforma MPLS−DS referentes à

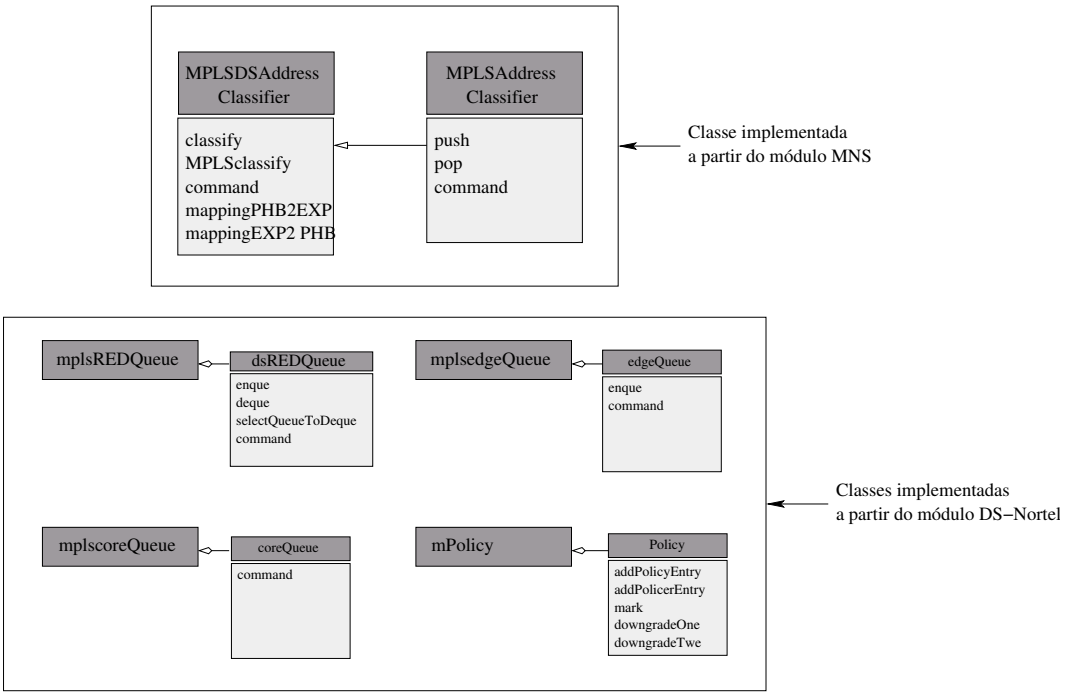

Figura 3.9: Classes do módulo MPLS−DS.

implementação DS−Nortel. As duas primeiras são comuns, tanto para os roteadores de borda como de núcleo, e as duas últimas somente são aplicadas nos roteadores de borda.

```
$fila addPHBEntry      $phb $fila $precedencia
$fila configQ          $fila $precedencia $minTh $maxTh $prob

$fila addPolicerEntry $policiador $codepoint_inicial $downgrade_codepoint
$fila addPolicyEntry  $origem $destino $policiador $codepoint_inicial $cir $cbs
```

3.5.3 Implementação do mecanismo de mapeamento

O mecanismo de mapeamento de classes de serviço se encarrega de associar os pacotes de um fluxo que entram na plataforma a uma classe de serviço e um LSP. Este mapeamento deve ser especificado no contrato SLS estabelecido entre a rede cliente e a plataforma, como por exemplo, os pacotes das classes de melhor esforço e AF serão transportados em um E−LSP.

Nos roteadores de borda da rede MPLS−DS existem estruturas de dados contendo as in-

formações necessárias para realizar o mapeamento, que são utilizadas pelos classificadores dos roteadores para poder selecionar os pacotes que serão diferenciados. A API *insert-mapping* é a que se encarrega de informar o mapeamento PHB ↔ EXP para o roteador LSR-DiffServ de borda e armazena essa informação na estrutura de dados.

Como mostramos no código abaixo, o roteador de borda LSR1 realizará o mapeamento da classe com PHB 100 para a classe 110 codificada no campo EXP do cabeçalho MPLS. Os outros parâmetros desta API correspondem ao tipo de LSP e a seu identificador, que transportará as classes mapeadas no LSP com *E−LSPid* 2000.

```
set phb 100
set exp 110
set tipo ELSP
set lspid 2000
$ns at $tempo "[$LSR1 get-module "MPLS/DiffServ"] \
               insert-mapping $phb $exp $tipo $lspid"
```

O tratamento dado aos pacotes é inferido a partir do campo EXP. Essa funcionalidade é implementada, tanto para os roteadores LSR-DiffServ de borda como de núcleo, nas classes *mplsedgeQueue* e *mplscoreQueue*. As funções relacionadas ao policiamento do tráfego são especificadas na classe *mPolicy* e realizadas somente pelos roteadores de borda. A figura 3.10 exemplifica o mapeamento das classes de serviço.

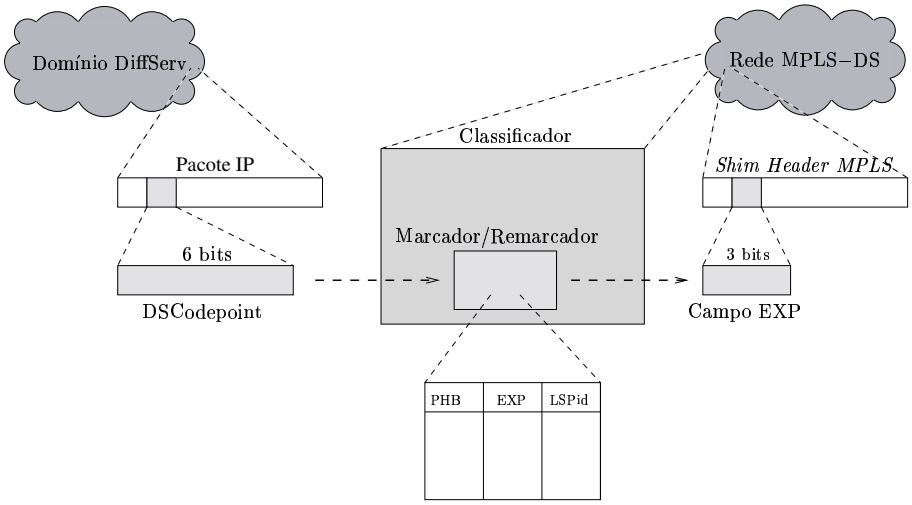

Figura 3.10: Mapeamento de classes de serviço.

3.5.4 Implementação da arquitetura de políticas

Como foi descrito na Seção 3.4, o modelo de informação da arquitetura de políticas utilizado pela plataforma MPLS−DS corresponde ao modelo de políticas de três camadas formado pelo PDP, PEP e o repositório de políticas. Também foi mencionado que o *bandwidth broker* e o PDP são elementos que podem estar implementados no mesmo objeto já que ambos se complementam. Por esse motivo, a partir de agora, passaremos a chamar esses dois elementos como sendo o *Bandwidth Broker*.

O *bandwidth broker* é um programa independente ao simulador *NS*. O PEP, por ser um elemento de configuração do roteador está implementado no próprio simulador *NS*. E o repositório de políticas é um arquivo preenchido manualmente.

A conexão do *bandwidth broker* com a rede MPLS−DS é realizada através da arquitetura CORBA [7] (*Common Object Request Broker Architecture*), que nos fornece a inter-operabilidade entre o *bandwidth broker* e o simulador *NS*. A motivação para utilizar CORBA se deve a:

migração para um ambiente real, a implementação do *bandwidth broker* é independente ao simulador *NS*. A arquitetura CORBA nos oferece a possibilidade de utilizar a implementação do *bandwidth broker* em uma rede real como mecanismo encarregado de gerenciar os recursos de rede disponíveis.

heterogeneidade, permite que o *bandwidth broker* seja implementado em qualquer linguagem, possibilitando que o acesso ao *bandwidth broker*, através de CORBA, seja transparente em relação à linguagem utilizando-se de um mapeamento em IDL.

A comunicação entre o *bandwidth broker* e o PEP é realizada por meio de chamadas de métodos dos objetos destes elementos. Basicamente, são trocadas as mensagens de requisição, decisão e de notifição do estado dos enlaces da rede. Na continuação, descreveremos as implementações do *bandwidth broker*, do PEP e do repositório de políticas.

Bandwidth Broker

O *bandwidth broker* foi implementado na linguagem iTcl [26]. O mapeamento de CORBA para a linguagem iTcl é fornecido pelo pacote de desenvolvimento COMBAT [30], através do uso da DII (*Dynamic Invocation Interface*) e a DSI (*Dynamic Skeleton Interface*). Este pacote permite a criação de *scripts* em ambos os lados, cliente e servidor, de uma comunicação CORBA.

A implementação do *bandwidth broker* realizada conta com os seguintes componentes:

- Interface de aplicação
- Interface de comunicação intra-domínio
- Interface com algoritmos de roteamento
- Interface com o repositório de políticas
- Interface com o PDP

A interface usuário/aplicação possibilita que os usuários ou as aplicações realizem requisições ao *bandwidth broker* por recursos de rede. A interface de comunicação intra-domínio possibilita a comunicação com os PEPs para notificação do estado da rede ou uma requisição de alocação de recursos RAR (*Resource Allocation Requests*). A interface com os algortimos de roteamento é ativada para encontrar uma nova rota com banda disponível para criação de um novo LSP, segundo uma requisição ou ativação de uma política. A interface com o PDP e a interface do repositório de políticas são utilizadas para decidir quando e para quem criar um LSP ou reconfigurar as filas para uma determinada classe, levando em conta os contratos SLSs e os recursos de rede disponíveis.

O *bandwidth broker* é composto por dois *daemons*: *pdpserver* e *rmanager*, que devem ser instanciados antes da execução de um *script* de simulação. O *pdpserver* recebe as requisições dos PEPs, e o *rmanager* tem a função de gerenciar as requisições por recursos de rede monitorando o arquivo de configuração da rede. Todos os objetos PEP criados para cada nó LSR-DiffServ se registram no *pdpserver* fornecendo sua IOR, associando-a com a identificação do PEP.

A figura 3.11 mostra como foi implementada a arquitetura de políticas da plataforma MPLS−DS e também mostra os componentes do *bandwidth broker*.

PEP

O PEP foi implementado na classe PEP criada no simulador *NS*. Esta classe possui métodos que são evocados, através do ORB CORBA, pelo *bandwidth broker* para que este configure os recursos de rede segundo a informação recebida, como por exemplo, configurar a fila 0 para a classe de serviço preferencial, ou criar uma rota explícita para a classe de serviço x. Cada PEP se registra no *bandwidth broker* fornecendo sua IOR.

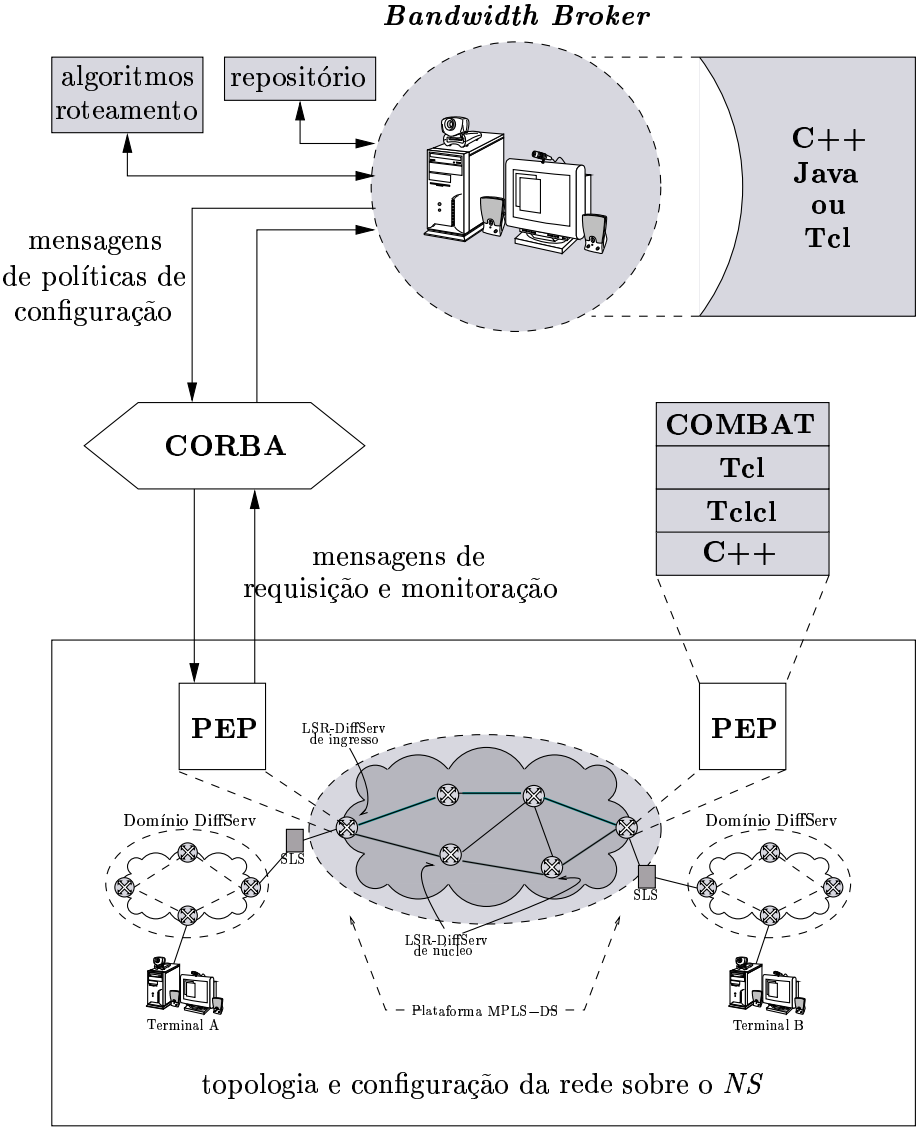

Figura 3.11: Implementação da arquitetura de políticas e o *Bandwidth Broker*.

Repositório de políticas

O repositório de políticas é um arquivo que contém as políticas que serão aplicadas no domínio da rede. Este arquivo é acessado diretamente pelo *bandwidth broker* para tomar alguma decisão quando este receber pedido de reserva de recursos para uma determinada classe de serviço.

3.5.5 Contrato SLS e políticas de configuração

O contrato SLS é traduzido em parâmetros de QoS de configuração e em regras de políticas, possibilitando a configuração dos elementos da rede para que um determinado fluxo de tráfego, que está classificado e marcado, receba tratamento diferenciado.

Os parâmetros de QoS de configuração indicam como serão configurados os roteadores LSR-DiffServ e como os pacotes marcados receberão tratamento diferenciado. Esses parâmetros de configuração são recebidos pelos PEPs dos roteadores da plataforma e estes configuram os elementos da rede.

As políticas são empregadas quando ocorrer alguma requisição de algum PEP devido a pré-configuração ou em resposta ao monitoramento da rede, como por exemplo: mediante a monitoração dos pacotes da classe preferencial se detetou que o atraso máximo que os pacotes desta classe podem sofrer foi ultrapassado, portanto os PEPs notificarão ao *bandwidth broker* (PDP) que o atraso foi ultrapassado e que alguma atitude deve ser tomada. O *bandwidth broker* por sua vez faz uma consulta ao servidor de políticas para decidir o que fazer com a requisição recebida.

3.6 Resumo

Neste capítulo, formalizamos a proposta desta dissertação referente à validação de políticas de configuração de rede por meio da utilização da plataforma MPLS−DS. Descrevemos os componentes e a maneira como foi implementada a plataforma. Também, foi realizada uma breve introdução ao simulador de redes *NS*.

No capítulo seguinte, descreveremos as políticas de configuração que foram implementadas e testadas. Também apresentamos os cenários simulados e os resultados obtidos nas simulações com o objetivo de validar a implementação da plataforma.

Capítulo 4

Aplicação da plataforma

A plataforma MPLS−DS tem como objetivo validar políticas de configuração que podem ser empregadas no *backbone* da rede Internet. Para poder realizar as simulações construímos a plataforma como especificado no Capítulo 3. Neste capítulo, dedicaremo-nos a validar a implementação da plataforma por meio das simulações realizadas.

Inicialmente, apresentamos o contexto onde será empregada a plataforma. Descreveremos as políticas e os recursos utilizados (algoritmos para descoberta de rotas) para poder realizar a configuração da rede. A seguir, descreveremos as características da rede simulada, apresentaremos a topologia adotada, os geradores de tráfego utilizados e as classes de serviço oferecidas pela rede. Para finalizar, descreveremos as simulações realizadas e apresentaremos os resultados obtidos utilizando a plataforma.

4.1 Contexto de atuação

Imaginemos que o *backbone* da rede Internet está configurado com os componentes da plataforma MPLS−DS e que redes DiffServ utilizam-na como uma rede de trânsito, como visto na figura 4.1. Entre essas redes e a plataforma, existe um contrato SLS que terá de ser respeitado, e um conjunto de políticas de configuração para cada rede cliente, interpretadas pelo *bandwidth broker*.

Dentro de cada uma das redes DiffServ os pacotes marcados com algum PHB são diferenciados. A idéia principal é que os pacotes pertencentes a uma classe de serviço, vindo das redes DiffServ, continuem sendo tratados, dentro do domínio da plataforma, por uma classe de serviço equivalente à do domínio DiffServ.

Por exemplo, um tráfego de uma aplicação de voz que se origina na rede DiffServ A com destino à rede DiffServ B terá que ser diferenciado dentro da plataforma, devido as características deste tipo de tráfego, para que sua transmissão e a qualidade da aplicação não sejam prejudicadas.

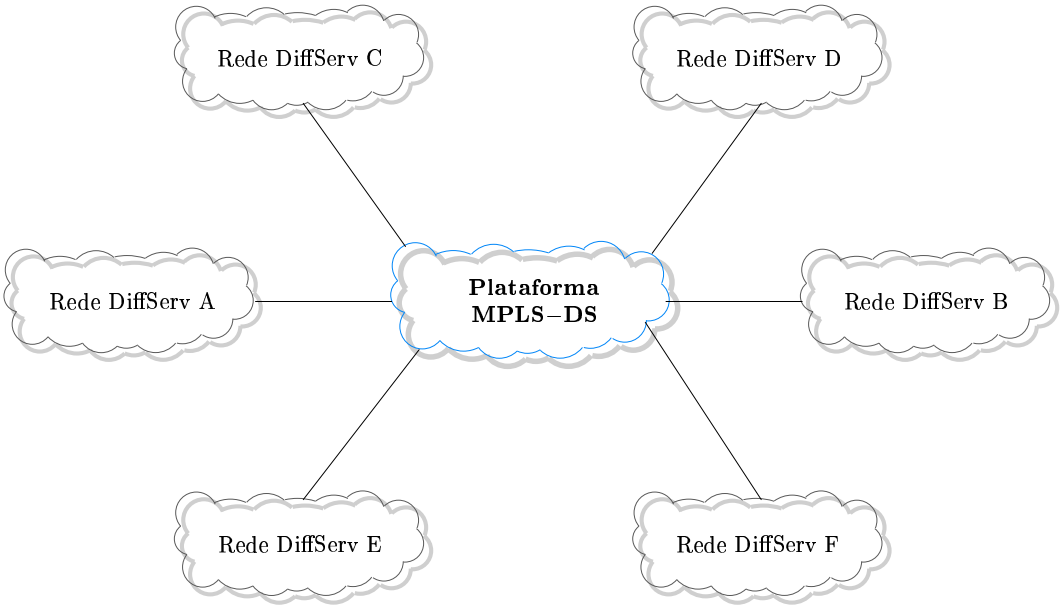

Figura 4.1: Contexto de atuação da plataforma MPLS−DS.

Além do *backbone* estar configurado com características de uma rede MPLS−DS, provendo a diferenciação de classes de serviço e a associação dessas classes a LSPs (E−LSPs e L−LSPs), é necessário que os pacotes do tráfego de voz (pacotes associados a uma classe de serviço) sejam monitorados, durante a execução da aplicação, para que sua transmissão não seja prejudicada quando ocorrer algum congestionamento na rede ou mesmo quando ocorrer a queda de algum enlace. Essa tarefa de monitoramento da rede é realizada pelo *bandwidth broker*.

O *bandwidth broker* recebe temporariamente notificações dos PEPs de cada roteador, informando o estado da rede, como também possui informações inferidas do contrato SLS e das políticas estabelecidas para uma classe de serviço ou de forma geral para uma rede cliente. Com todas essas informações, o *bandwidth broker* pode tomar decisões de como tratar algum fluxo de pacotes que está associado a uma classe de serviço.

Imaginemos, por exemplo, que por meio do contrato e as políticas definidas para uma aplicação de voz entre as redes DiffServ A e B, o *bandwidth broker* deverá configurar os roteadores da plataforma para tratar os pacotes dessa aplicação com o PHB EF, e levar em conta que o atraso

máximo de transferência dentro do *backbone* deverá ser 200 milisegundos. Se o atraso for excedido, o *bandwidth broker* poderá decidir criar um LSP com banda disponível para que o fluxo de pacotes da aplicação de voz não seja atrasado.

Políticas de configuração

Notemos que, pelo exemplo dado acima, entre a rede cliente e a plataforma MPLS−DS existe um contrato relacionado ao tráfego gerado pela aplicação de voz. Esse contrato diz como os pacotes devem ser tratados dentro do *backbone*. Além disso, por meio da monitoração realizada pelo *bandwidth broker*, é necessário que este tome decisões quando ocorrer algum congestionamento na rede que prejudique os pacotes do tráfego de voz ou quando o contrato especificado for violado, como citado no exemplo acima: o atraso da transmissão exceda o atraso máximo contratado.

As decisões tomadas pelo *bandwidth broker* estão baseadas nas regras de políticas inferidas do servidor de políticas PDP. No caso do exemplo, há uma política relacionada ao atraso para o fluxo de pacotes da aplicação de voz que é ativada se o atraso da transmissão for excedido. Neste caso, a política adotada é criar um LSP com nova rota para o fluxo de pacotes da aplicação de voz, com a finalidade de evitar que o atraso da transmissão seja excedido.

As políticas que são testadas e implementadas atualmente na plataforma são: a ***criação de LSPs*** com novas rotas, a ***configuração dos pesos das filas*** que estão associadas às classes de serviço e o ***mapeamento das classes de serviço em um LSP***. Obviamente, outras políticas podem ser implementadas, como por exemplo: aumentar a porcentagem da banda reservada para uma determinada classe de serviço, mudar o mecanismo de escalonamento, etc.

Decidimos que a política de criação de LSPs será ativada quando o atraso relacionado a uma classe de serviço for ultrapassado. A política de configuração das filas é ativada caso a porcentagem do descarte dos pacotes pertencentes a uma classe de serviço seja excedida. A política de mapeamento das classes de serviço em um LSP é realizada implicitamente quando um novo LSP é criado, pois a política de criação de LSPs é disparada caso o tráfego pertencente a uma classe X esteja sendo prejudicado, fazendo com que o novo LSP seja mapeado com o tráfego da classe X.

Todas as requisições feitas ao *bandwidth broker* informam o identificador do LSP pelo qual o fluxo medido está sendo transportado. Sabendo isto, o *bandwidth broker* averigua que roteadores necessitam ser configurados. Os parâmetros de configuração que o *bandwidth broker* retorna para os PEPs de cada roteador são mostrados na tabela 4.1.

Quando ativada a política de configuração de filas, o *bandwidth broker* retorna a *fila* que

será configurada e o *peso* que ela terá em cada roteador do LSP. A configuração é realizada instantâneamente.

Política	Parâmetros				Ativação
Configuração da fila	*fila*	*peso*			instantâneo
Mapeamento de classes	*lspid*	*phb*	*exp*		instantâneo
Criação do LSP	*rota*	*fec*	*lspid*	*phb*	100ms

Tabela 4.1: Parâmetros de configuração retornados pelas políticas.

A política de mapeamento de classes de serviço para um LSP é ativada quando um LSP é criado. O *bandwidth broker* retorna a informação para os PEPs dos roteadores de borda, indicando que os pacotes pertencentes à classe de serviço *phb* serão mapeados na classe de serviço *exp* e transitarão pelo LSP com identificador *lspid* dentro do domínio da plataforma MPLS−DS. Esta configuração também é executada instantâneamente.

O *bandwidth broker* quando toma a decisão de criar um LSP retorna para o roteador de ingresso a *rota* por onde o novo LSP será construído. O LSP criado tem um identificador *lspid* e é instalado para a FEC *fec* com classe de serviço *phb*. A criação do LSP é realizada instantâneamente mas a instalação dos fluxos no LSP é realizada após 100 milisegundos.

Recursos utilizados

No caso de se ativar a política de criação de LSPs, o *bandwidth broker* deverá possuir um mecanismo para que ele possa descobrir uma rota disponível dentro do *backbone* para poder criar o LSP com a restrição de suportar o tráfego associado. Esta tarefa é realizada com a utilização de algoritmos de descoberta de rotas, como por exemplo: banda residual, MIRA, etc. Estes algoritmos são implementados e comparados em [24].

Atualmente, o *bandwidth broker* utiliza o algoritmo de banda residual para descobrir uma rota disponível na rede e assim poder criar o LSP. Para poder descobrir a rota, é necessário fornecer, para o procedimento da banda residual, a banda requerida Y para que o algoritmo descubra uma rota com banda Y disponível.

Com a utilização da plataforma MPLS−DS é possível gerenciar os recursos de rede para os fluxos de pacotes pertencentes a uma classe de serviço levando em conta políticas de configuração estabelecidas para cada uma das classes.

A seguir, descreveremos a configuração da rede utilizada para simular cenários onde pode ser aplicada a plataforma.

4.2 Configuração da plataforma

Nesta seção descreveremos a configuração que foi empregada na plataforma para realizar os testes desejados. Apresentaremos a topologia da rede, descreveremos as fontes de tráfego utilizadas e as classes de serviço suportadas pela plataforma.

4.2.1 Topologia da rede

A figura 4.2 mostra a topologia da rede simulada. O *backbone* é formado por sete roteadores LSR-DiffServ enumerados de 6 a 12. Os roteadores 6, 7, 8, 11 e 12 são LSR-DiffServ de borda. Os roteadores LSR-DiffServ de núcleo são os nós 9 e 10.

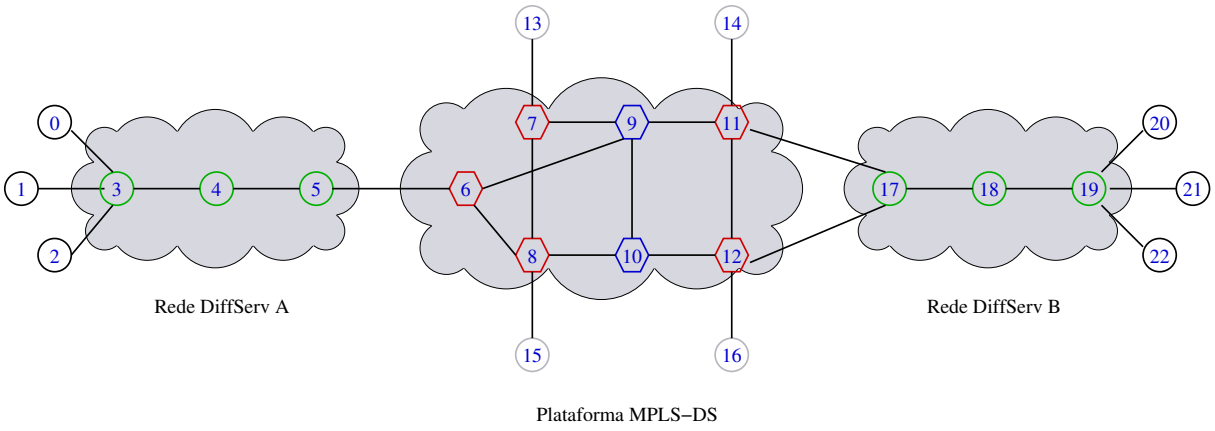

Figura 4.2: Topologia da rede simulada.

Os roteadores de borda possuem funções para: realizar o controle de admissão, a classificação e marcação de pacotes, o policiamento do tráfego marcado, realizar o mapeamento das classes de serviço e associá-las a um determinado LSP ou a distintos LSPs . Os roteadores do núcleo se encarregam de enviar os pacotes para o próximo nó e tratá-los segundo o PHB associado. Este comportamento é inferido a partir do campo EXP do cabeçalho MPLS.

A rede DiffServ A é formada pelos roteadores 3, 4 e 5. Os roteadores 3 e 5 são os roteadores de borda e realizam a classificação e o policiamento do tráfego que entra à rede. Analogamente,

a rede DiffServ B é formada pelos roteadores 17, 18 e 19, onde os roteadores de borda são o 17 e 19. Os roteadores 4 e 18 são roteadores do núcleo das redes DiffServ cuja única função é tratar de forma diferenciada os pacotes marcados e encaminhá-los para o próximo nó. O objetivo de se criar especificamente esses dois domínios é o de verificar a marcação e o policiamento do tráfego que está associado às classes de serviço dessas redes.

Os roteadores 13, 14, 15 e 16 simulam roteadores de egresso de outras redes DiffServ. O objetivo destes roteadores é o de gerar tráfego de fundo para que os enlaces do *backbone* fiquem saturados e se produza o congestionamento.

A função dos nós de origem 0, 1 e 2, é o de gerar tráfego para os nós de destino 20, 21 e 22, respectivamente. O tráfego gerado em cada um desses nós possui diferentes características, portanto o encaminhamento dentro das redes DiffServ e no *backbone* deve ser diferenciado.

Capacidade dos enlaces e filas

Os enlaces das redes DiffServ e do *backbone* podem transmitir até 6,912Mbs.

São utilizadas quatro filas para poder oferecer três classes de serviços (BE[1], AF e EF), sendo que para a classe AF são utilizadas duas filas devido à precedência de descarte. Cada uma das filas pode armazenar até 1000 pacotes, independente do tamanho do pacote. O tipo de fila utilizado para as classes é a fila do tipo *DropTail* e o escalonador utilizado é um escalonador WRR (*Weigthed Round Robin*). Cada classe de serviço está associada a uma fila, sendo que a fila associada para a classe EF possui maior prioridade ou peso. As filas associadas à classe preferencial AF têm peso inferior em relação à fila da classe EF.

4.2.2 Fontes de tráfego

Especificamos três tipos de fontes geradoras de tráfego: uma para o serviço de melhor esforço, outra para gerar tráfego que simule uma aplicação de voz e uma terceira fonte para simular o tráfego correspondente a um serviço preferencial (como por exemplo: tráfego de uma aplicação de comércio eletrônico). Na continuação descreveremos as características de cada uma dessas fontes.

[1]BE, *Best Effort* ou melhor esforço.

Tráfego de melhor esforço

Os pacotes do tráfego de melhor esforço são gerados por diversas fontes, sendo que todas elas estão configuradas com os mesmos valores. O tamanho dos pacotes deste tipo de tráfego é de 512 bytes. O tipo de gerador de tráfego utilizado corresponde a uma aplicação de tráfego exponencial e a taxa de geração dos pacotes é de 1.728Mbs o que equivale a 25% da capacidade dos enlaces. A tabela 4.2 mostra as características do tráfego de melhor esforço.

A classe associada a este tipo de tráfego é a de melhor esforço (BE) e os pacotes pertencentes a esta classe são marcados com o *codepoint* 0. Estas fontes são utilizadas para gerar tráfego de interferência no *backbone* para poder gerar congestionamento nos enlaces e poder forçar a que o *bandwidth broker* tome decisões.

Tamanho do pacote	512 bytes
Tipo de tráfego	Exponencial
Taxa de geração de pacotes	1.728Mbs

Tabela 4.2: Tráfego de melhor esforço.

Tráfego de voz

A aplicação de voz é simulada por uma fonte de tráfego CBR (*Constant Bit Rate*) que gera pacotes de 66 bytes a cada 20 milisegundos. Esta aplicação é executada durante toda a simulação. São simuladas 115 chamadas telefônicas. O modelamento de geração de pacotes para esta aplicação é adotado segundo [12] que especifica como deve ser realizada a transmissão de voz sobre IP (VoIP).

Os pacotes que pertencem a esta aplicação serão marcados com o *codepoint* 20 que é o PHB associado à classe EF na plataforma. A característica da classe EF é garantir baixo atraso, baixa taxa de descarte e baixa variação do atraso. Os pacotes desta classe são tratados preferencialmente em relação aos pacotes dos outros tráfegos gerados. A tabela 4.3 mostra as características do tráfego de voz.

Tamanho do pacote	66 bytes
Tipo de tráfego	CBR
Taxa de geração de pacotes	22ms

Tabela 4.3: Tráfego da aplicação de voz.

Tráfego preferencial

O tráfego preferencial está associado a uma aplicação que gera pacotes exponencialmente com 512 bytes de tamanho, cuja taxa de geração de pacotes é de 1.728Mbs. Os pacotes pertencentes à classe preferencial (AF) são marcados com os *codepoints* 30 e 31 para a re-marcação do tráfego, e a transmissão deles é priorizada em relação aos pacotes de melhor esforço. A característica de transmissão deste tipo de tráfego é que os pacotes pertencentes a esta classe não podem ter uma taxa alta de descarte, mas em contrapartida podem sofrer atrasas altos. A tabela 4.4 mostra os parâmetros adotados para o tráfego preferencial.

Tamanho do pacote	512 bytes
Tipo de tráfego	Exponencial
Taxa de geração de pacotes	1.728Mbs

Tabela 4.4: Tráfego preferencial.

4.2.3 Classes de serviço fornecidas

As classes de serviço suportadas pela plataforma MPLS−DS são: a classe de melhor esforço (BE), a classe expressa (EF) e a classe preferencial (AF). A tabela 4.5 mostra os *codepoints* utilizados para as classes de serviço. Não necessariamente, os *codepoints* usados na plataforma MPLS−DS são os mesmos das redes DiffServ.

Classes	*Codepoint*
Classe de melhor esforço (BE)	0
Classe preferencial (AF 1-1)	30
Classe preferencial (AF 1-2)	31
Classe expressa (EF)	20

Tabela 4.5: Classes suportadas pela plataforma.

Os pacotes que pertencem à classe de melhor esforço são os primeiros a serem descartados. Já os pacotes associados à classe de serviço EF são tratados com prioridade, mas se eles violarem o perfil de tráfego estabelecido serão descartados diretamente. Se os pacotes que pertencem à classe preferencial violarem o perfil de tráfego, eles serão re-marcados com um *codepoint* com prioridade inferior, mas não são descartados diretamente.

4.2.4 Contrato SLS estabelecido

Para os pacotes pertencentes à aplicação de voz foi estabelecido o seguinte contrato de serviço: a transmissão dos pacotes deste tipo de tráfego deve ter baixo atraso e pouca perda de pacotes. O atraso máximo tolerado é de 250 milisegundos e a porcentagem de pacotes perdidos não deve exceder de 20% do total de pacotes enviados.

Atraso máximo	250ms
Descarte máximo	20%

Tabela 4.6: Contrato para a aplicação de voz.

Para os pacotes pertencentes ao tráfego preferencial foi estabelecido o seguinte contrato: a transmissão dos pacotes não deve ter um atraso maior do que 4 segundos e a porcentagem de descarte não deverá exceder 20 % do total de pacotes enviados.

Atraso máximo	4s
Descarte máximo	20%

Tabela 4.7: Contrato para o tráfego preferencial.

As tabelas 4.6 e 4.7 mostram os parâmetros dos contratos estabelecidos para o tráfego de voz e preferencial, respectivamente.

Os pacotes pertencentes ao tráfego de melhor esforço não receberão tratamento diferenciado e deverão ser encaminhados quando houver recursos de rede disponíveis. Isto significa que os pacotes pertencentes a este tipo de tráfego serão descartados com maior freqüência e a fila para esta classe será servida com baixa prioridade.

4.3 Simulação dos cenários e resultados

A simulação realizada compreende três fases. A primeira se refere à criação de um E−LSP para os fluxos gerados pela rede DiffServ A com destino à rede DiffServ B. A segunda fase se refere à criação de L−LSPs entre as demais redes DiffServ, com a intenção de gerar concorrência pelos recursos de rede e forçar o *bandwidth broker* a tomar decisões baseando-se nas políticas de configuração de rede. A terceira fase é dedicada a descrever as notificações recebidas e ações tomadas pelo *bandwidth broker*. O tempo de duração da simulação é de 180 segundos, devido a que o arquivo gerado pela simulação vai ficando grande a medida que o tempo de simulação vai aumentando.

Nas redes DiffServ A e B os pacotes da aplicação de voz, do tráfego de melhor esforço e preferencial, são marcados com distintos *codepoints*, portanto recebem distintos PHBs. Dentro dessas redes os pacotes da aplicação de voz são marcados com o *codepoint* 10. Os pacotes do tráfego preferencial são marcados com os *codepoints* 15 e 16 em caso da re-marcação, e os pacotes do tráfego de melhor esforço com o *codepoint* 0. A aplicação de voz recebe o PHB da classe EF, os pacotes do tráfego preferencial recebem o PHB da classe AF e os pacotes do tráfego de melhor esforço recebem o PHB da classe BE.

Os *codepoints* utilizados para as classes de serviços oferecidas dentro da plataforma são especificados na tabela 4.5. Os tráfegos gerados entre as redes DiffServ A e B são os seguintes: A aplicação de voz é originada no nó 0 tendo como destino o nó 20. Os pacotes gerados a partir do nó 1 com destino ao nó 21 pertencem à classe de melhor esforço. O fluxo de pacotes da classe preferencial é gerado no nó 2 com destino no nó 22. A figura 4.3 mostra as fontes de tráfego geradas a partir da rede DiffServ A com destino a rede DiffServ B.

Paralelamente, outros tráfegos de melhor esforço e de aplicações de voz são originados a partir dos nós 13, 14, 15 e 16 (considerados roteadores de ingresso/egresso de outras redes DiffServ), conforme descritos posteriormente nas fases dois e três.

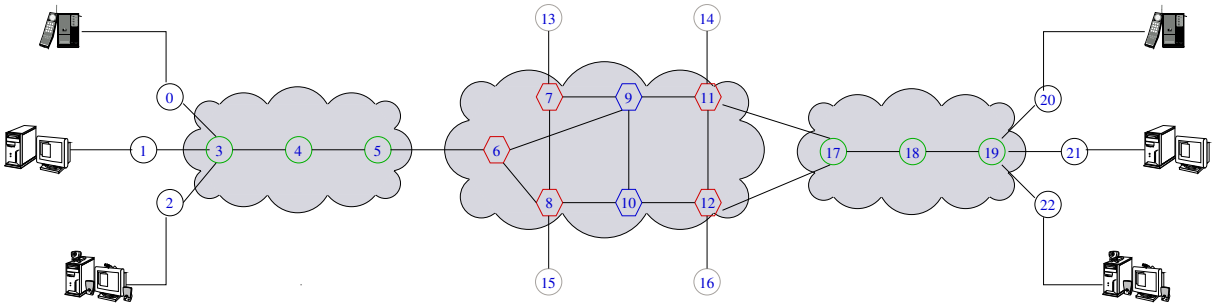

Figura 4.3: Tráfegos gerados.

Primeira fase

Inicialmente, o *backbone* é pré-configurado para transportar todos os fluxos de pacotes gerados por clientes da rede DiffServ A com destino à rede DiffServ B no E−LSP 2000. Todos os fluxos, no início da simulação, pertencem à classe de melhor esforço, portanto são marcados com o *codepoint* 0. O E−LSP 2000, criado a partir do roteador LSR-DiffServ 6, segue a rota $[6, 9, 11]$, escolhida explicitamente por ser a mais curta.

Posteriormente, no roteador de borda LSR-DiffServ 6, a tabela de mapeamento de classes de serviço possuirá a configuração como mostrado na figura 4.4. A tabela de mapeamento mostrada na figura é interpretada da seguinte maneira: para os pacotes que chegam marcados no roteador de borda com algum *condepoint*, procura-se na tabela de mapeamento se esses pacotes deverão ser diferenciados ou não dentro do *backbone*. Caso positivo, eles serão associados a uma classe de serviço e a um LSP. Por exemplo: os pacotes da aplicação de voz são marcados pela rede DiffServ A com o *codepoint* 10; quando chegarem ao roteador de ingresso do *backbone* esses pacotes serão mapeados para a classe com *codepoint* 20 e esse fluxo de pacotes será transportado no E−LSP 2000. Analogamente, no roteador de egresso LSR-DiffServ 11, será realizado o mapeamento das classes de serviço para a rede *downstream*.

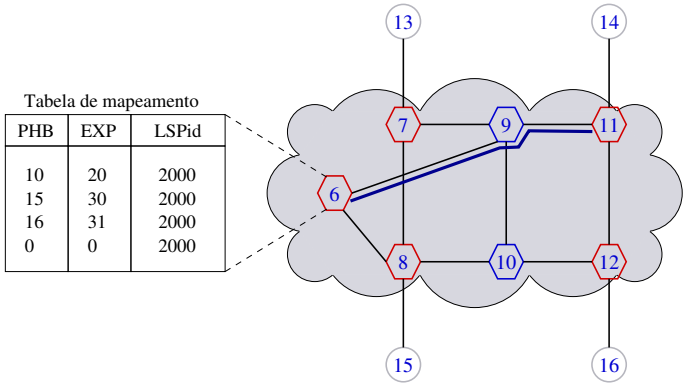

Figura 4.4: Rota seguida pelo E−LSP 2000.

Nas redes DiffServ é utilizado o campo TOS ou COS do cabeçalho do pacote, dependendo da versão do protocolo IP, para inferir o PHB do pacote. Já nas rede MPLS−DS o PHB do pacote é inferido a partir do campo EXP, como foi descrito no capítulo anterior, Seção 3.5.3.

Ação tomada pelo *bandwidth broker*

À medida que o tráfego do *backbone* aumenta e os enlaces começam a saturar-se, o *bandwidth broker* decidirá realizar alguma ação para evitar o congestionamento e evitará que os contratos de serviço sejam violados baseando-se nas políticas de configuração. Portanto, por meio do contrato, o *bandwidth broker* decidirá associar cada fluxo a uma classe de serviço e esses fluxos continuarão sendo transportados pelo E−LSP 2000. A diferenciação dos pacotes é realizada utilizando-se uma fila para cada PHB e dando pesos para cada uma das filas. O peso inicial para a fila do tráfego

de voz é 10, o peso para a fila do tráfego preferencial é 2 e o peso da fila para o tráfego de melhor esforço é 1.

A configuração das filas é ativada devido ao descarte em excesso dos pacotes da aplicação de voz, notificado pela monitoração do *bandwidth broker*. Quando o *bandwidth broker* percebe que os pacotes do tráfego de voz estão sendo descartados com uma taxa acima do estipulado no contrato, ele ativará a política para configurar as filas para cada classe de serviço. A configuração das filas pode ser realizada até quatro vezes. Caso a taxa de descarte continue crescendo, uma nova política é considerada. A nova política adotada será a de criar um LSP, com nova rota, para o tráfego prejudicado.

O gráfico 4.5, mostra os instantes em que a política de configuração das filas é disparada para os pacotes do trafego de voz e preferencial devido ao excesso da taxa de descarte. Como podemos ver no gráfico, a política de configuração de filas para o tráfego de voz é disparada nos instantes 2.6s, 10.6s e 11.1s. A porcentagem de descarte nesses instantes são 28.59%, 21.45% e 68.36%, respectivamente. Para o tráfego preferencial, esta política é disparada nos instantes 6.09s, 15.1s, 15.6s e 16.1s onde a porcentagem de descarte nesses instantes são 31.25%, 29.04%, 20.09%, e 47.54%, respectivamente.

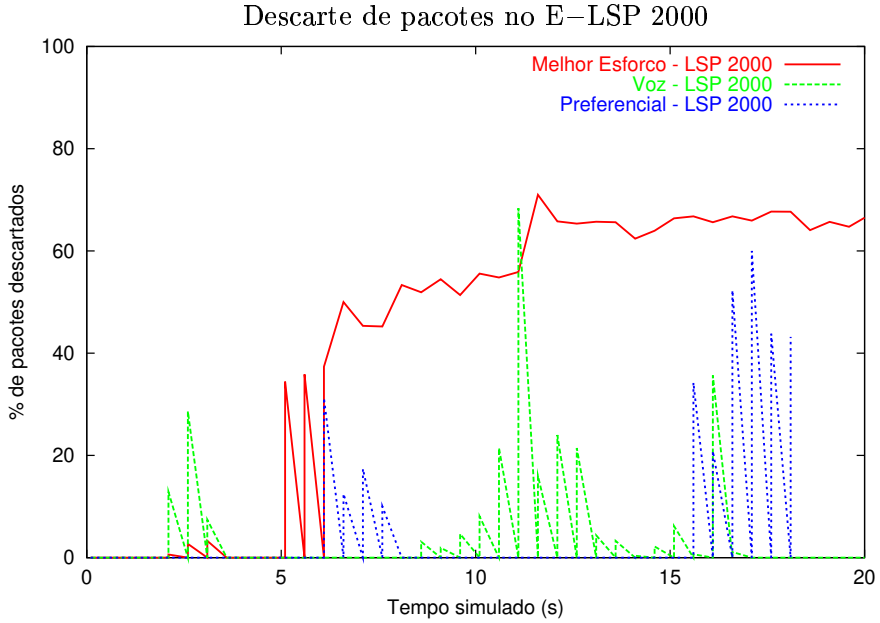

Figura 4.5: Monitoração do descarte no E−LSP 2000.

Notemos que no caso do tráfego preferencial, a política para reconfiguração das filas, para a

classe AF, é disparada quatro vezes. A partir da quinta notificação de excesso de descarte a política a ser adotada é a de criar um novo LSP para esse tráfego. A atuação dessa política é apresentada na última fase. Os dados de descarte listados foram resgatados do arquivo de *log* do *bandwidth broker*.

O gráfico 4.6, mostra o atraso sofrido nas filas pelos pacotes das classes transportadas no E−LSP 2000 durante os 20s iniciais da simulação. Podemos notar que nesse período não foi ativada a política de criação de LSPs, diretamente, pois o atraso da transmissão dos pacotes não foi maior do que o atraso máximo contratado.

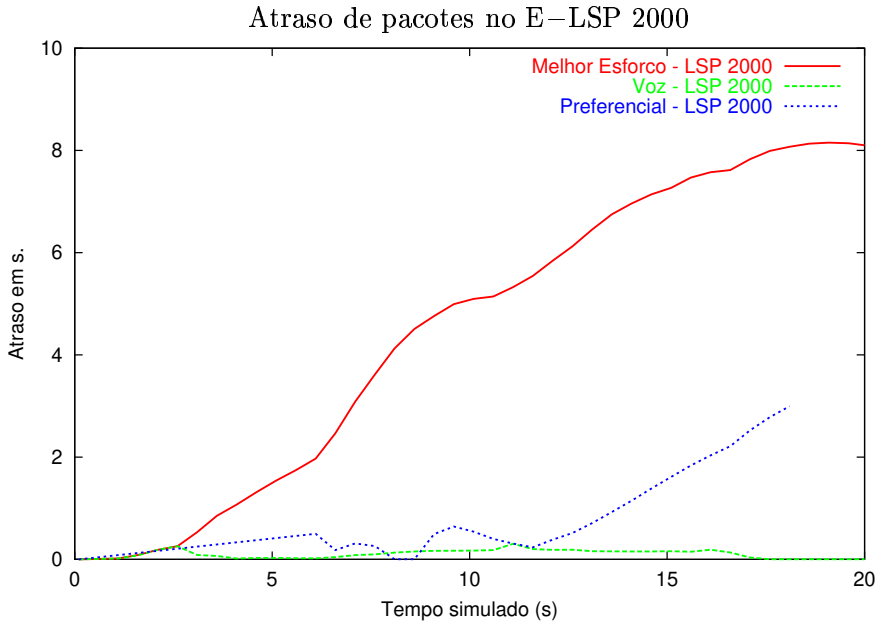

Figura 4.6: Monitoração do atraso no E−LSP 2000.

Segunda fase

Além do E−LSP 2000, foram criados três L−LSPs entre outras redes DiffServ, para gerar concorrência dos recursos de rede entre os pacotes de distintas origens mas pertencendo à mesma classe de serviço.

Os LSPs criados foram: o L−LSP 1000 seguindo a rota $[7, 9, 11]$, o L−LSP 1001 seguindo a rota $[7, 8, 10, 12]$, ambos criados no instante t=10s da simulação, e o L−LSP 1002 seguindo a rota $[8, 10, 12]$ criado no instante t=80s. A figura 4.7 mostra o caminho seguido pelos L−LSPs criados.

Parte do arquivo de configuração é listado abaixo mostrando os LSPs que foram criados.

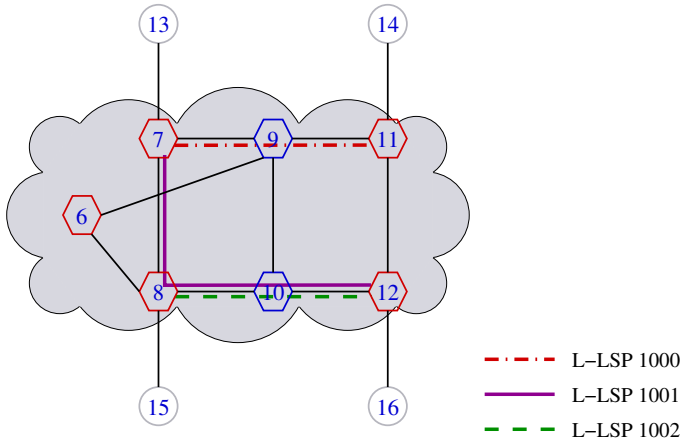

Figura 4.7: L−LSPs criados.

Esse arquivo é interpretado pelo *bandwidth broker* quando se realiza a pré-configuração da rede.

```
%LSP_CREATION:
LSR1 fonedst 6_9_11 2000 20 0.10
LSR2 alt1    7_9_11 1000 20 10.00
LSR2 alt3    7_8_10_12 1001 20 10.00
LSR3 alt3    8_10_12 1002 20 80.00

%LSP_MONITORING:
LSR1 2000 0.10
LSR2 1000 10.00
LSR2 1001 10.00
LSR3 1002 80.00
```

Os L−LSPs criados transportam pacotes de aplicações de voz geradas entre as redes DiffServ. O L−LSP 1000 é criado entre os LSR-DiffServ de borda 13 e 14, o L−LSP 1001 é criado entre os roteadores LSR-DiffServ 13 e 16 e o L−LSP 1002 entre os roteadores LSR-DiffServ 15 e 16.

À medida que a simulação evolui e o tráfego na rede aumenta, o *bandwidth broker* recebe requisições e informações de monitoração informando-o sobre o estado da rede. O *bandwidth broker* agirá para evitar que o tráfego associado às classes de serviço seja prejudicado e os contratos de serviço assinados não sejam violados. As ações tomadas pelo *bandwidth broker* são descritas na próxima fase.

Terceira fase

Os gráficos de monitoração, apresentados na figura 4.8 e 4.9, mostram os instantes em que o *bandwidth broker* é notificado quando o descarte e o atraso de pacotes para o tráfego de voz e preferencial são violados.

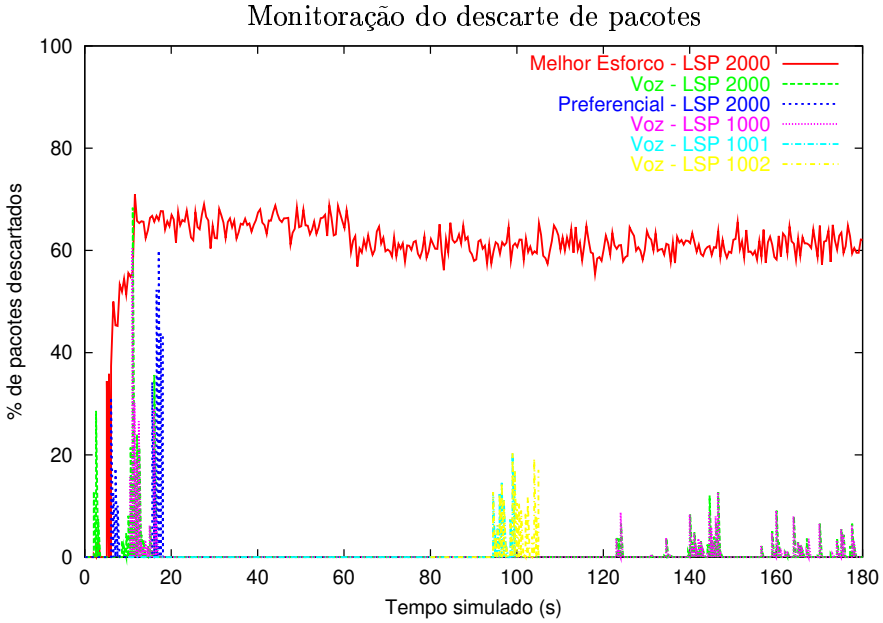

Figura 4.8: Descarte de pacotes nos LSPs pré-configurados.

Como podemos ver no gráfico 4.8, os pacotes que estão associados à classe de melhor esforço (traço vermelho) são os mais prejudicados devido a que eles possuem uma baixa prioridade de transferência em relação ao demais pacotes. Para os tráfegos associados as classes expressa e preferencial são respeitados os contratos como foram especificado nas tabelas 4.6 e 4.7, respectivamente.

Da mesma maneira, como vemos no gráfico 4.9, no caso do atraso para os tráfegos pertencentes às classes expressa e preferencial o contrato assinado é respeitado, passando-se a criar novos LSPs para os fluxos que estão sendo prejudicados. Para os tráfegos de voz (traços ciano e amarelo) são criados novos LSPs, no instante 100 da simulação aproximadamente, já que o atraso estipulado no contrato foi excedido, ultrapassando os 250ms contratados. Para o tráfego preferencial (traço azul) também é respeitado o contrato, mas não é por causa do atraso excedido que é criado um novo LSP para ele e sim por ter-se ativado pela quinta vez a política de reconfiguração da fila associada a este tráfego, que obedece à notificação do excesso do descarte de pacotes para o tráfego

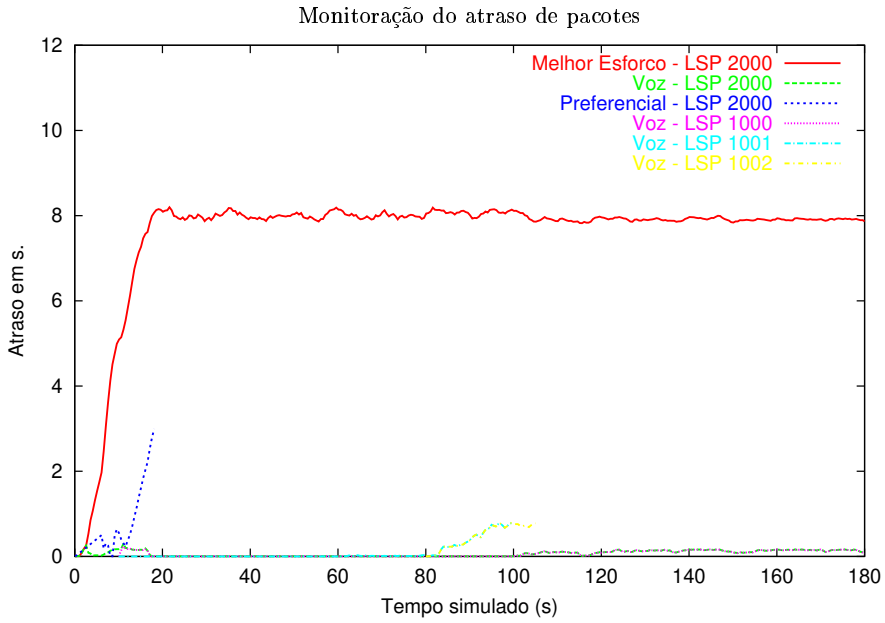

Figura 4.9: Atraso de pacotes nos LSPs pré-configurados.

preferencial. O tráfego de melhor esforço é o mais prejudicado sofrendo maior atraso em relação aos demais tráfegos, já que este tipo de tráfego possui baixa prioridade de transferência.

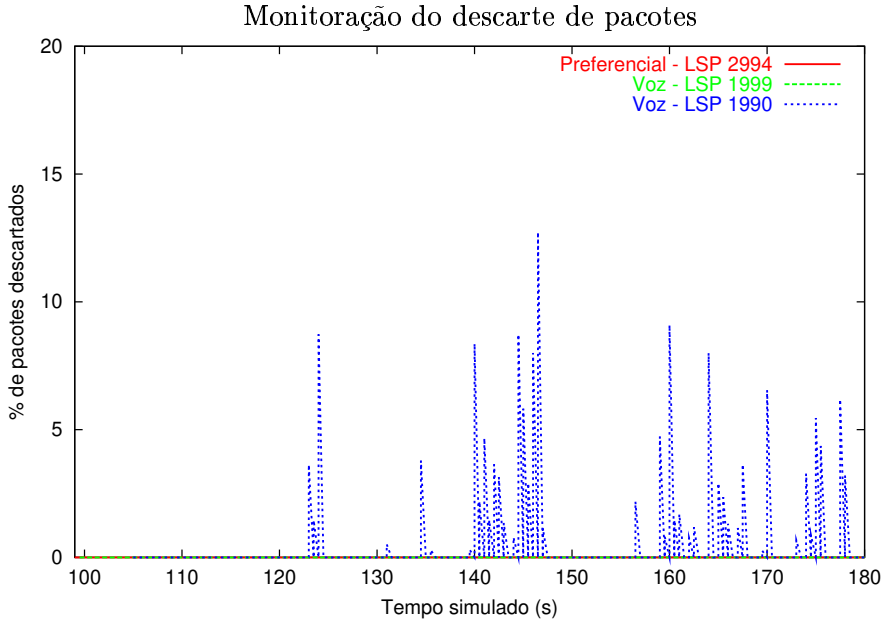

Figura 4.10: Descarte de pacotes nos LSPs criados pelo *bandwidth broker*.

Os gráficos das figuras 4.10 e 4.11 mostram o descarte e o atraso dos pacotes nos LSPs criados pelo *bandwidth broker* em resposta às requisições recebidas dos PEPs e da monitoração realizada nos LSPs.

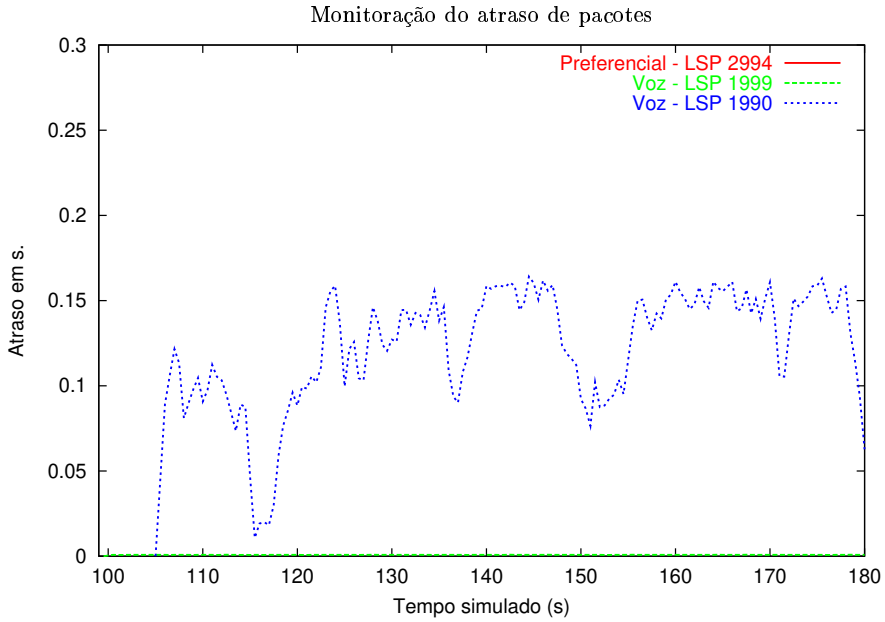

Figura 4.11: Atraso de pacotes nos LSPs criados pelo *bandwidth broker*.

Como vemos nos gráficos acima, tanto o descarte como o atraso não são mais violados para os tráfegos de voz e preferencial, portanto nenhuma ação é tomada pelo *bandwidth broker*.

Ações tomadas

Quando o L−LSP 1000 é instalado, transportando tráfego de voz, começa a competir pelos recursos de rede com o tráfego similar do E−LSP 2000, isto leva o *bandwidth broker* a aumentar o peso da fila associada à classe EF. Por outro lado, o tráfego preferencial e o tráfego de melhor esforço são prejudicados.

O *bandwidth broker* receberá uma notificação indicando que o descarte de pacotes do tráfego preferencial está sendo excedido, como a notificação de descarte é realizada pela quinta vez, a política para a criação de um LSP será ativada. A rota seguida pelo novo E−LSP é $[6, 8, 10, 12]$. O mapeamento da classe de serviço AF (tráfego preferencial) foi realizado para o E−LSP 2994 implicitamente.

No instante t=81s é iniciado o tráfego de voz do E−LSP 1002. Devido à concorrência por recursos de rede entre os tráfegos de voz dos E−LSPs 1001 e 1002, o atraso da transmissão desses pacotes será prejudicado, como pode ser visto nas figuras 4.12 e 4.13, aumentando o tempo de atraso sofrido nas filas.

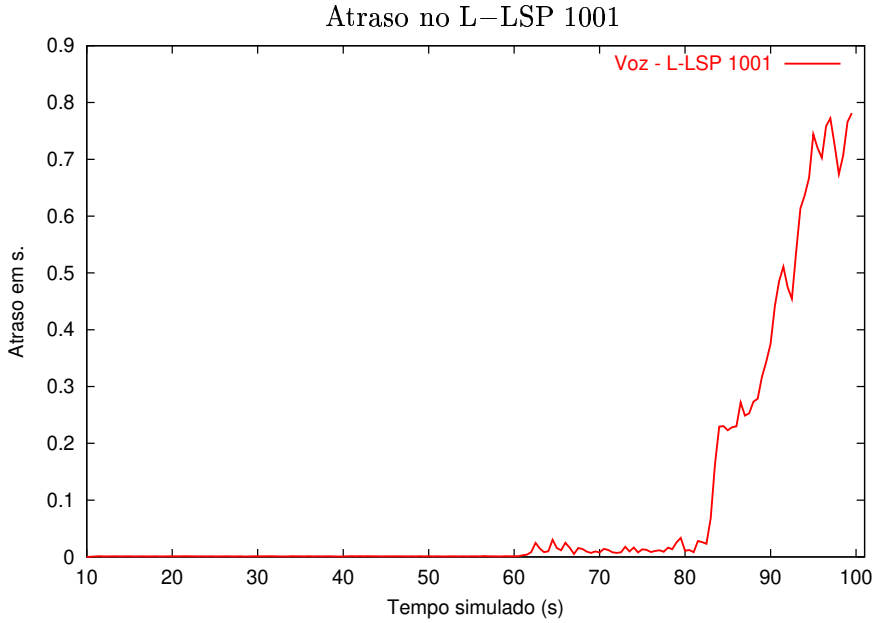

Figura 4.12: Atraso para os pacotes de voz no L−LSP 1001.

Devido ao aumento do atraso da transmissão dos pacotes do tráfego de voz nos LSPs 1001 e 1002, o *bandwidth broker* criará dois L−LSPs para que o contrato de serviço não seja violado. Os L−LSPs criados pelo *bandwidth broker* possuem os identificadores 1998 e 1988, as rotas seguidas por eles são $[7-9-10-12]$ e $[8-7-9-11-12]$ respectivamente. Logo em seguida, o *bandwidth broker* desativa os L−LSPs 1001 e 1002.

Resultados

Os gráficos a seguir são comparativos e mostram o que aconteceria se os pacotes fossem roteados normalmente pelo protocolo IP. Além disso, nenhuma consideração de QoS é levada em conta. A medição realizada é do atraso da transmissão dos pacotes para: a aplicação de voz, tráfego preferencial e dos pacotes pertencentes ao tráfego de melhor esforço gerados entre as redes DiffServ A e B.

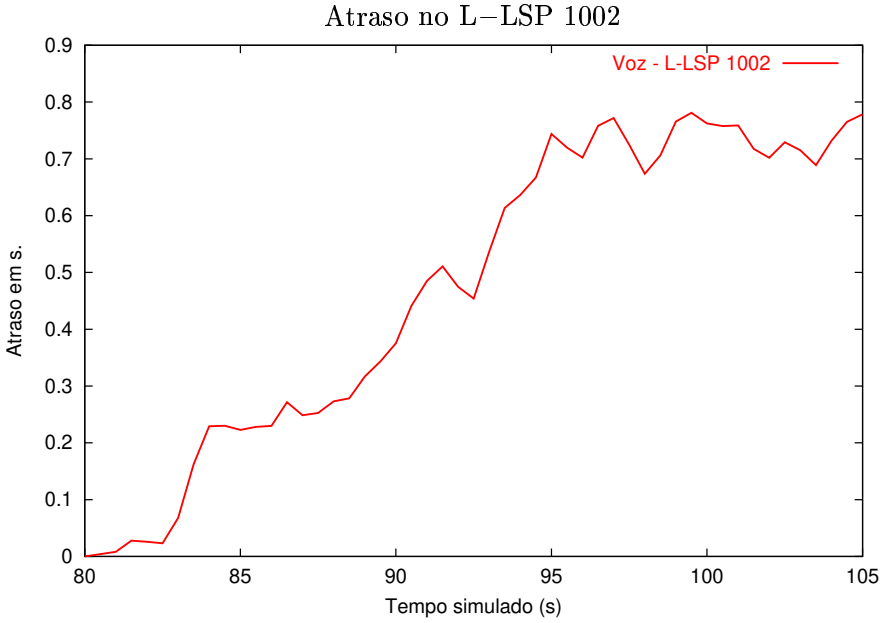

Figura 4.13: Atraso para os pacotes de voz no L−LSP 1002.

A figura 4.14 mostra o atraso da transmissão dos tráfegos de voz, preferencial e melhor esforço. Como constatamos no gráfico, os pacotes das três aplicações são tratados da mesma maneira e são roteados pelo mesmo caminho. Os pacotes pertencentes ao tráfego de voz sofrem altos atrasos prejudicando as aplicações deste tipo.

O tratamento dado aos pacotes das três aplicações é similar pois na Internet clássica não se tem a diferenciação de classes de serviço, portanto os atrasos de transmissão nas três aplicações são similares.

Na figura 4.15 é mostrada a porcentagem de descarte dos pacotes dos tráfegos de voz, preferencial e melhor esforço. Como vemos no gráfico, a porcentagem de descarte dos pacotes do tráfego de voz é alto, prejudicando a qualidade da aplicação. Estes pacotes são mais descartados devido a taxa em que os dados da aplicação de voz são gerados. De igual forma, a qualidade do tráfego preferencial também é prejudicada devido à alta taxa de descarte de pacotes. As aplicações associadas a este tipo de tráfego necessitam de que seus dados sejam entregados com a menor perda possível.

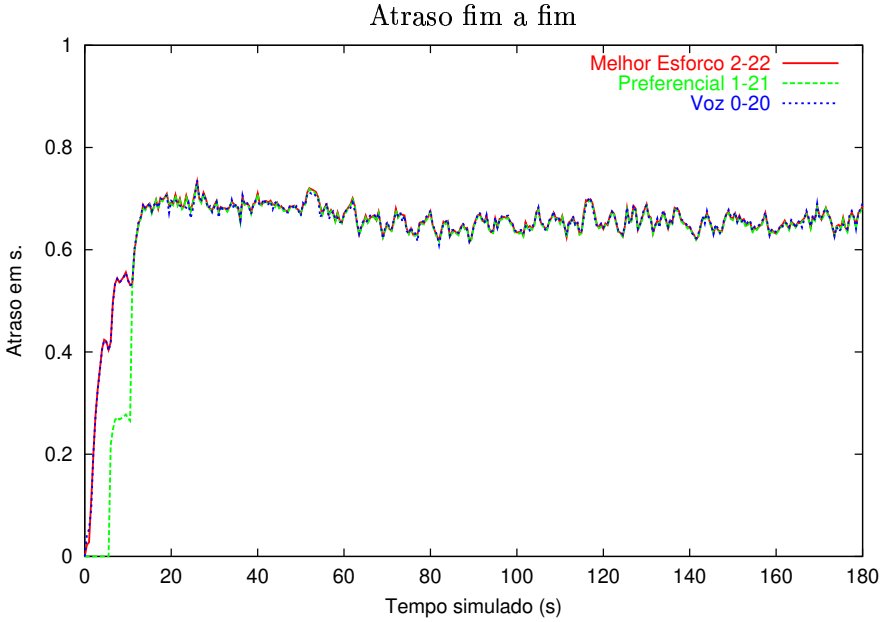

Figura 4.14: Medição do atraso no *backbone*.

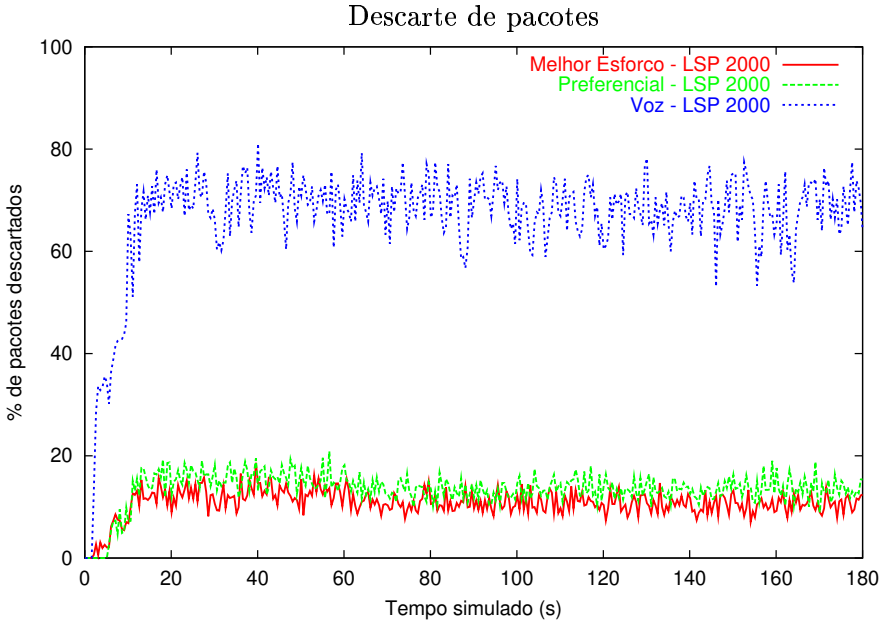

Figura 4.15: Medição do descarte de pacotes no *backbone*.

Os gráficos mostrados nas figuras 4.16 e 4.17 foram obtidos utilizando a plataforma como uma infra-estrutura para poder prover ao núcleo da rede características de QoS e engenharia de

tráfego mediante a utilização do *bandwidth broker* que gerencia os recursos de rede, monitora a rede e verifica o cumprimento das políticas de configuração.

A figura 4.16 mostra o atraso da transmissão do tráfego de voz no *backbone*. Como vemos no gráfico, o atraso da transmissão dos pacotes deste tráfego trata de ser amenizado pelo *bandwidth broker*. Inicialmente, no intervalo de 0 a 20ms, o *broker* recebe notificações sobre o excesso do atraso dos pacotes deste tráfego ultrapassando o estipulado no contrato associado à aplicação de voz. Tomando em conta as informações recebidas, ele passa a re-configurar o peso da fila associada a classe de serviço EF, a partir deste instante o tráfego de voz será priorizado em relação aos outros tráfegos, não violando mais o contrato de serviço assinado.

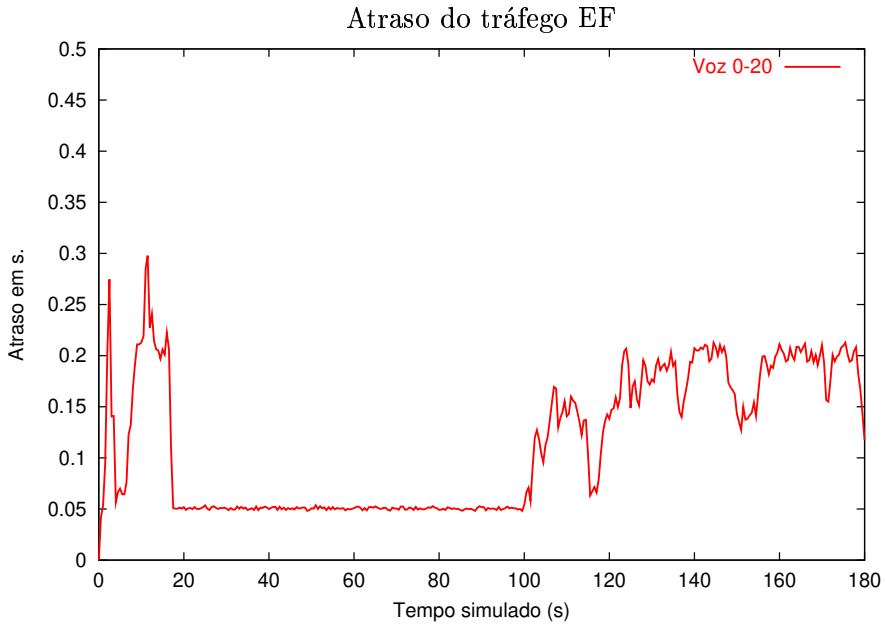

Figura 4.16: Atraso fim a fim - Tráfego de voz.

A figura 4.17 mostra o atraso da transmissão do tráfego preferencial e de melhor esforco no *backbone*.

De maneira análoga, o *bandwidth broker* cumpre o contrato de serviço para o tráfego preferencial, pois não ultrapassa o atraso de transferência deste tipo de tráfego, estipulado no contrato de serviços para este tipo de tráfego. No caso do tráfego de melhor esforço, nenhuma ação é tomada pelo *bandwidth broker* já que este tipo de tráfego não está associado a um contrato de serviço a ser cumprido pelo *broker*.

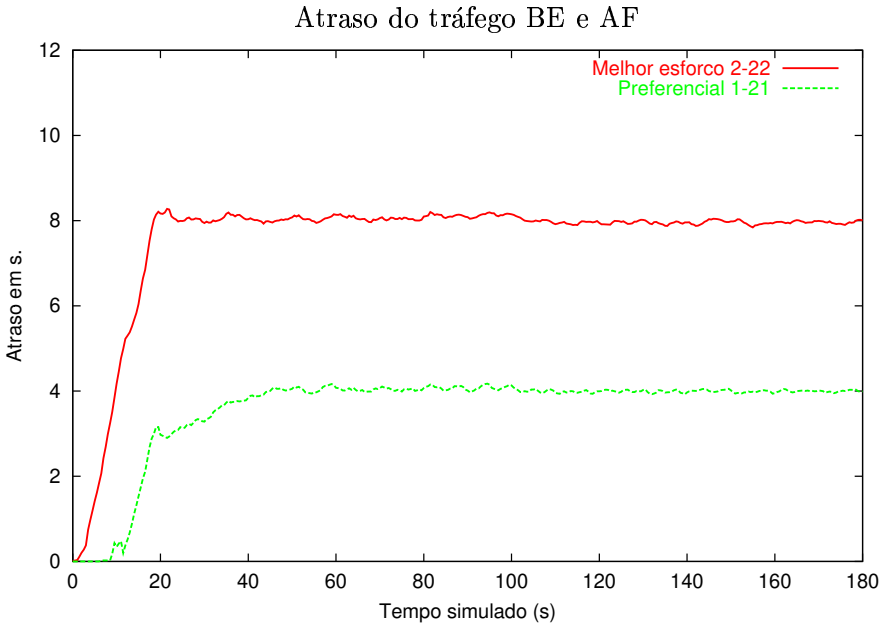

Figura 4.17: Atraso fim a fim - BE e AF.

Comparando os resultados obtidos nas figuras 4.14, 4.16 e 4.17, podemos ver que para os tráfegos de voz e preferencial se obteve um melhor resultado com a utilização da plataforma proposta. Além disso, passa-se a respeitar os contratos assinados com as redes clientes, já que o *bandwidth broker* evita violar o atraso da transferência de pacotes para os tráfegos de voz y preferencial.

O resultado mostrado na figura 4.18 está relacionado à porcentagem de descarte dos tráfegos de voz, preferencial e melhor esforço com a utilização da plataforma MPLS−DS no *backbone* da rede.

Como vemos no gráfico acima, o tráfego de melhor esforço é o mais prejudicado, devido a que este tráfego não esta associado a nenhum contrato de serviço. Por outro lado, os tráfegos de voz e preferencial são monitorados pelo *bandwidth broker* e os contratos de serviço associados a cada tráfego são respeitados, já que as porcentagens de descarte contratadas não são ultrapassadas.

Comparando os resultados obtidos nos figuras 4.15 e 4.18 vemos que existe uma melhora significativa com a utilização da plataforma MPLS−DS em relação ao descarte de pacotes dos tráfegos de voz e preferencial.

Pelas simulações realizadas, verificamos que com a utilização da plataforma no *backbone* da rede Internet, aumentamos o desempenho da rede, passando a utilizar caminhos que não eram utilizados. Por outro lado, a qualidade da transmissão de aplicações é melhorada por meio da

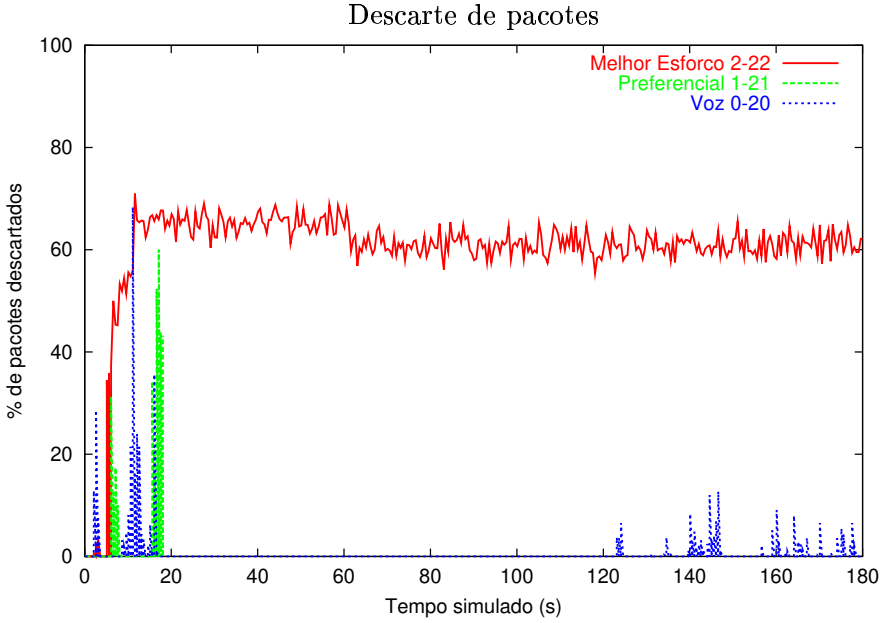

Figura 4.18: Descarte de pacotes das classes EF, AF e BE.

diferenciação dos pacotes das distintas aplicações, pois se associa cada tipo de tráfego a uma classe de serviço e se respeitam os contratos de serviço para cada tipo de tráfego.

Capítulo 5

Conclusão

A contribuição deste trabalho está no desenvolvimento da plataforma MPLS−DS como ferramenta para permitir a validação de políticas de configuração para uma rede, realizada por meio de simulações. Acreditamos que com a utilização desta ferramenta poderemos implementar variadas políticas de configuração que podem ser levadas em conta em uma rede para configurá-la segundo as suas necessidades e respeitando os contratos assinados com os clientes. As políticas implementadas são validadas no simulador *NS*.

O *bandwidth broker*, junto ao PDP, são ferramentas importantes para o gerenciamento de recursos de rede. A utilização da arquitetura CORBA na comunicação entre esta ferramenta e a rede simulada permite a sua exportação para um ambiente de rede real, no qual são reaproveitadas as políticas implementadas e avaliadas no simulador.

O simulador *NS* mostrou ser uma ferramenta muito importante que pode ser utilizado para a validação de diversas implementações, principalmente pela flexibilidade que oferece na mudança de códigos referentes a protocolos ou à implementação de novas tecnologias, devido ao conceito de orientação de objetos. Várias universidades e centros de pesquisa têm utilizado o simulador para implementar e testar as funcionalidades de novas tecnologias propostas. Prova disto, são as implementações da arquitetura das redes MPLS e da arquitetura das redes DiffServ [1, 29], respectivamente.

A utilização das redes MPLS com suporte à diferenciação de serviços fornece a engenharia de tráfego e a priorização de tráfegos associados a classes de serviço, otimizando a utilização dos recursos da rede e diferenciando os fluxos de pacotes pertencentes às classes de serviço fornecidas no domínio da plataforma. Estas características fornecem a QoS com uma granularidade mais fina para a rede Internet.

É importante ressaltar que o MPLS necessita de um mecanismo para descobrir rotas por onde se possam construir LSPs com banda disponível. A descoberta de uma rota com banda disponível para uma determinada classe de serviço foi realizada utilizando o algoritmo de banda residual, como implementado em [24].

Tanto clientes como provedores de serviço Internet, necessitam saber se a qualidade de serviço contratada entre eles está sendo respeitada ou não. Esse interesse pode ser atendido pela monitoração do *bandwidth broker* através de simulações.

Como foi apresentado no final da Seção 4.3 concluímos que, com a utilização da plataforma no *backbone* rede Internet, obtém-se ganhos significativos relacionados à qualidade de serviço dada às diversas aplicações que utilizam a Internet como meio de transporte, considerando que cada uma das aplicações está associada a uma classe de serviço.

A arquitetura de políticas permite-nos mapear políticas de alto nível das empresas em parâmetros de configuração específicos para os dispositivos da rede. Além disso, a utilização de políticas automatiza a tarefa de gerenciamento de recursos de rede.

Trabalhos futuros

O gerenciamento das filas realizado neste trabalho não abordou as filas que utilizam o algoritmo RED e suas variações. Os parâmetros de configuração destes algoritmos, gatilhos de ativação e valores das probabilidades de descarte podem ser manipulados por meio de políticas. Esta abordagem poderia permitir que os parâmetros de configuração considerassem não somente o estado atual da fila, mas talvez o comportamento das filas anteriores no caminho do LSP ou mesmo ter um histórico de configuração das filas para poder reconfigurá-las de forma mais precisa. Além disso, políticas para o estabelecimento desses parâmetros podem ser implementadas como mencionado em [8].

Conforme discutido na Seção 3.5, a implementação do *bandwidth broker* e do PDP foi realizada independentemente ao simulador *NS*. A comunicação com os PEPs implementados no simulador é realizada através da arquitetura CORBA. Essa independência faz com que a implementação possa migrar para uma rede real para ser utilizada na gerência de recursos levando em conta as políticas implementadas.

A comunicação com o repositório de políticas pode ser melhorada utilizando o LDAP [3] para a troca de informações com o *bandwidth broker* e o PDP. A troca de mensagens entre o PDP

e os PEPs pode ser realizada utilizando-se o protocolo COPS [10]. Por outro lado, a formalização da representação de políticas também deve ser considerada, como apresentada no âmbito da IETF [37].

Outros algoritmos para a descoberta de rota podem ser utilizados, como por exemplo: o MIRA e suas variações, propostos e analisados em [24].

Na especificação do projeto Qbone [27], existe um ítem que trata da comunicação entre *bandwidth brokers*, ativada quando surge uma requisição de uma aplicação que atravessa vários domínios (cada domínio possui um *bandwidth broker*) para realizar a reserva de recursos. Simulações dentro deste contexto também podem ser realizadas, mas para isto são necessárias realizar algumas modificações na implementação do *bandwidth broker*. Além disso, o *bandwidth broker* pode gerenciar a autenticação, autorização e tarifação para um determinado cliente que está associado a uma classe de serviço. Essas funcionalidades também devem ser implementadas dentro do *bandwidth broker*.

Com o objetivo de dar maior robustez à plataforma MPLS−DS, a implementação do *bandwidth broker* está sendo migrada para a linguagem C++. A implementação realizada em [3] conta com: um PDP, implementado em Java, e um repositório de políticas que utiliza o LDAP para a comunicação com *bandwidth broker* e o PDP. Essa implementação pode ser suportada pela plataforma, pois as interfaces implementadas para esses elementos são acessadas através de CORBA, viabilizando a comunicação entre eles. Portanto, a integração das implementações do *bandwidth broker*, do PDP e do repositório de políticas poderá ser realizada.

Referências Bibliográficas

[1] Gaeil Ahn and Woojik Chun. Overview of MPLS Network Simulator: Design and implementation. http://flower.ce.cnu.ac.kr/ fog1/mns/index.htm, December 1999.

[2] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas AND. Ldp Specification. IETF, January 2001. RFC 3036.

[3] Marcos Antonio de Siqueira. Aplicação de uma Arquitetura de Políticas para gerência de redes MPLS e QoS. Tese de Mestrado em andamento, Departamento de Computação e Automação Industrial, Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas - Unicamp, 2001.

[4] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP tunnels. IETF, December 2001. RFC 3209.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. IETF, December 1998. RFC 2475.

[6] Mabia Cavalcante. Algoritmos de Balanceamento de Carga para Tráfego tipo Melhor Esforço em Redes IP/MPLS. Tese de Mestrado, Departamento de Computação e Automação Industrial, Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas - Unicamp, 2001.

[7] Object management group. http://www.omg.org.

[8] Leonardo Costa. Desenvolvimento de um Bandwidth Broker para a Arquitetura de Serviços Diferenciados. Tese de Mestrado, Instituto de Computação da Universidade Estadual de Campinas - IC Unicamp, 2001.

[9] DARPA. http://www.darpa.mil/.

[10] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. The COPS Common Open Policy Service Protocol. IETF, January 2000. RFC 2748.

[11] Mauricio Ferreira Magalhães e Eleri Cardozo. Qualidade de Serviço na Internet. Relatório Técnico, Universidade Estadual de Campinas DCA/FEEC/UNICAMP, Outubro 1999.

[12] Mauricio Ferreira Magalhães e Eleri Cardozo. Convênio de Cooperação Tecnológica Embratel/Unicamp - Aditivo Número 7 Relatorio Final. Relatório Técnico, Universidade Estadual de Campinas DCA/FEEC/UNICAMP, Julho 2000.

[13] A. Elwaid, C. Jin, S. Low, and I. Widjaja. MATE: MPLS Adaptive Traffic Engenineering. INFOCOM, 2001.

[14] Kevin Fall and Kannan Varadhan. *NS Notes and Documentation.* ISI, August 2000.

[15] Francois Le Faucheur, Liwen Wu, Bruce Dave, Shahram Davari, pasi Vaananen, Ram Krishnan, Pierrick Cheval, and Juha Heinanen. MPLS support of Differentiated Services. IETF, February 2001. Internet Draft, draft-ietf-mpls-diffserv-ext-08.txt.

[16] Dan Grossman. New Terminology for Diffserv. IETF, March 2001. Internet Draft, draft-ietf-diffserv-new-terms-04.txt.

[17] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. IETF, June 1999. RFC 2597.

[18] Geoff Huston. *Internet Performance Survival Guide.* Wiley Computer Publishing, 2000.

[19] IETF Home Page. http://www.ietf.org.

[20] Internet2 web site. http://www.internet2.edu.

[21] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. IETF, June 1999. RFC 2598.

[22] LBNL. http://www-nrg.ee.lbl.gov/.

[23] Tulius Lima. nsPLEx- Plataforma de Experimentação de Algoritmos de Engenharia de Tráfego MPLS no simulador NS. Tese de Mestrado em andamento, Departamento de Computação e Automação Industrial, Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas - Unicamp, 2001.

[24] Vinicius Lopes da Costa. Algoritmos de Roteamento. Tese de Mestrado em andamento, Departamento de Computação e Automação Industrial, Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas - Unicamp, 2001.

[25] Alex Marcelo Samaniego Guillén y Mauricio Ferreira Magalhães. Calidad de Servicio en el Backbone IP con DiffServ y MPLS. *Congreso Internacional Intercom 2000*, 2000.

[26] Michael J. McLennan. Incr-tcl - version 3.0 for Tcl/Tk 8.0.3. http://www.tcltk.com/itcl/.

[27] Rob Neilson, Jeff Wheeler, Francis Reichmeyer, and Susan Hares. A Discussion of Bandwidth Broker Requirements for Internet2 Qbone Deployment. Technical report, Internet2 Qbone BB Advisory Council, 1999.

[28] Network Simulator. http://www.isi.edu/nsnam/ns, 1995.

[29] Peter Pieda, Jeremy Ethridge, Mandeep Baines, and Farhan Shallwani. *A Network Simulator Differentiated Services Implementation.* Nortel Networks, July 2000. Open IP, Nortel Networks.

[30] Frank Pilhofer. Pagina do pacote de desenvolvimento combat. http://www.informatik.uni-frankfurt.de/ftp/Tcl, Maio 2000.

[31] Qbone home page. http://qbone.internet2.edu.

[32] Quality of service forum. http://www.qosforum.com/.

[33] Braden R., Clark D., and Shenker S. Integrated services in the internet architecture: an overview. IETF, June 1994. RFC 1633.

[34] Raju Rajan, Dinesh Verma, Sanjay Kamat, Eyal Felstaine, and Shai Herzog. A policy framework for integrated and differentiated services in the internet. *IEEE Network*, 1999.

[35] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, and A. Conta. MPLS Label Stack Encoding. IETF, January 2001. RFC 3032.

[36] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. IETF, January 2001. RFC 3031.

[37] Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, and B. Moore. Policy qos information model. IETF, November 2001. Internet Draft, draft-ietf-policy-qos-info-model-04.txt.

[38] Tequila project. http://www.ist-tequila.org.

[39] UC Berkeley. http://www.mash.cs.berkeley.edu/.

[40] USC/ISI. http://www.isi.edu/.

[41] VINT Project. http://www.isi.edu/nsnam/vint/.

[42] The Need for QoS. QoSforum.com, 1999.

[43] Quality of Service Differentiated Services and Multiprotocol Label Switching. Ericsson Australia, March 2000.

[44] Xerox Park. http://www.parc.xerox.com/.

[45] Xipeng Xiao. *Providing Quality of Service in the Internet.* PhD thesis, Michigan State University, 2000.