

Universidade Federal de Uberlândia - UFU
Faculdade de Computação - FACOM

Configuração Dinâmica de Classes DiffServ e Provisão de QoS para Aplicações Multimídia

Autor: Rayner Gomes Sousa

Orientador: Prof. Dr. Luís Fernando Faina

Dissertação de Mestrado apresentada à Faculdade de Computação como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação. Área de Concentração: **Redes de Computadores**.

Março de 2005
Uberlândia, MG - Brasil

FICHA CATALOGRÁFICA ELABORADA pelo
Sistema de Bibliotecas da UFU / Setor de Catalogação e Classificação

S725c

Sousa, Rayner Gomes - 1976,

Configuração Dinâmica de Classes DiffServ e Provisão
de QoS para Aplicações Multimídia / Rayner Gomes Sousa.
Uberlândia, 2005.

79f. : il.

Orientador: Luís Fernando Faina.

Dissertação (Mestrado) - Universidade Federal de Uberlândia,

Programa de Pós-Graduação em Ciência da Computação.

Inclui bibliografia.

1. Internet (Redes de Computadores) -Teses. 2. Sistemas Multimídia -
Teses. 3. Ciência da Computação - Teses. I. Faina, Luís Fernando.
II. Universidade Federal de Uberlândia. Programa de Pós-Graduação
em Ciência da Computação. III. Título.

CDU: 681.3.02 (043.3)

Configuração Dinâmica de Classes DiffServ e Provisão de QoS para Aplicações Multimídia

Autor: Rayner Gomes Sousa

Banca Examinadora

Prof. Dr. Luís Fernando Faina - Presidente - FACOM/UFU

Prof. Dr. Luiz Affonso H. G. de Oliveira - Membro Externo - DCA/UFRN

Prof. Dr. Jamil Salem Barbar - Membro Interno - FACOM/UFU

Prof. Dr. Pedro Frosi Rosa - Membro Interno - FACOM/UFU

Março de 2005
Uberlândia, MG - Brasil

Resumo

Sousa, Rayner Gomes, “Configuração Dinâmica de Classes DiffServ e Provisão de QoS para Aplicações Multimídia”. Dissertação de Mestrado - FACOM/UFU, Uberlândia, MG. Março/2005.

O crescimento contínuo da *Internet* é resultado da convergência dos vários mercados de comunicação (p.ex., telefonia, entretenimento e transmissão de dados) que utilizam-se de sua infra-estrutura para oferecer serviços. A qualidade de serviço das aplicações é influenciada por esse crescimento e a maioria delas trabalha com mídias contínuas, como áudio e vídeo, que são altamente sensíveis ao estado da rede, portanto, demandam que a rede lhes dê garantias de desempenho.

Qualidade de Serviço (QoS) é a capacidade da rede, de fornecer garantias às aplicações. Aplicações utilizam-se de parâmetros de QoS associados a sua camada para expressar seus desejos. Vários estudos são realizados para que a *Internet* possa prover QoS. Dentre esses merece destaque a Arquitetura DiffServ. Esta Arquitetura deixa alguns pontos em aberto que são base para proposta de um novo modelo, que em conjunto com Arquitetura DiffServ vão ao encontro para fornecer suporte completo e explícito para as exigências de QoS das aplicações.

Na especificação e no desenvolvimento do modelo, conceitos como orientação a objeto, sistemas distribuídos e controle de tráfegos são fundamentais. Segundo esses conceitos, são desenvolvidos os componentes: IQoS para tradução dos parâmetros da camada de aplicação para os de rede, TCD que configura todos os roteadores do domínio DiffServ, BB para gerenciar as classes DiffServ e atuar como controlador de admissão e o Ambiente de Gerenciamento que atua como interface entre o domínio DiffServ e o administrador do domínio.

Diante dessas considerações, este trabalho, inicialmente, apresenta os parâmetros de QoS das camadas de aplicação e de rede, as arquiteturas para adicionar QoS às redes IP, os motivos da adoção da Arquitetura DiffServ e também suas deficiências. Em seguida, um conjunto de componentes agregam funcionalidades à Arquitetura DiffServ para suprir essas deficiências. O modelo proposto é apresentado para avaliar as tecnologias para o desenvolvimento, controle de tráfego e análise de rede. Finalmente, uma seqüência de testes é realizada para demonstrar as funcionalidades e a importância dos componentes.

Palavras-chave: Arquitetura DiffServ, Controle de Tráfego no Linux, CORBA, Objetos Distribuídos e Qualidade de Serviço.

Abstract

Sousa, Rayner Gomes, “Dynamic Configuration of the DiffServ Classes and Provision of QoS for Multimedia Applications”. Master Thesis - FACOM/UFU, Uberlândia, MG. March 2005.

Internet’s continue growth is resulted of the convergence of several markets (p.ex. telephony, entertainment and transmission of data) that use your infrastructure for provision of the services. The quality of the services of the applications is influenced by this growth and most of them works with continuous medias as the audio and video that are highly sensitive with the state of the network, therefore, they demand that the network gives them guarantees of performance.

Quality of Service (QoS) is a capacity of the network that supplies guarantees to the applications. The applications use parameters of QoS associated to your layer to express your desires. Several studies are realized so that the Internet can provide QoS. One of the studies that are more detached is the DiffServ Architecture. This architecture leaves some points in open that are base for proposal of a new model, that together with the DiffServ are going to the encounter for supplying complete and explicit support for the demands of QoS of the applications.

In model’s specification and development, concepts as object oriented, distributed systems and control of traffics are fundamental. According to these concepts are developed this component: IQoS for translation of the parameters of the application layer for the one of network, the TCD that configures all the rotors of the DiffServ domain, the BB for manage the classes DiffServ and to act as admission controller and the Environment of Administration as interface among the domain DiffServ and the administrator of the domain.

Due to those considerations, this work initially presents the parameters of QoS of the application layer and of network, the architectures to add QoS to the IP networks, the reasons of Architecture DiffServ’s adoption and also your deficiencies. Soon after, a group of components aggregate functionalities to the DiffServ Architecture to supply your deficiencies. The proposed model is presented to evaluate the technologies for the development, control of traffic and network analysis. Finally, a sequence of tests is accomplished to demonstrate the functionalities and the importance of the components.

Keywords: Architecture DiffServ, Linux Network Traffic Control, CORBA, Distributed Objects and Quality of Services.

“São fúteis e cheias de erros as ciências que não nasceram da experimentação, mãe de todo o conhecimento.”

Leonardo da Vinci

“Conhecimento escondido em um canto escuro de um cérebro qualquer não é conhecimento; é orgulho.”

Ernani Mendeiros

Agradecimentos

DEUS, agradeço ao Senhor a oportunidade da realização deste trabalho. Neste mundo de tantos contrastes e desvios, precisamos de forças para que possamos continuar a evoluir na estrada da vida. Neste caminho tu és a força nos momentos em que acreditamos que não a temos mais, és a bússola para nos nortear em momentos de dúvida. Obrigado Senhor pela força e pela orientação.

Agradeço muito ao orientador Luís Fernando Faina pela direção e o acompanhamento deste trabalho e principalmente pelo profissional mais preparado que ele me ajudou a ser.

Ao Rossano, agradeço pela ajuda no desenvolvimento do trabalho, e principalmente por não conhecê-lo pessoalmente, a sua disponibilidade em me ajudar o caracteriza com um verdadeiro professor.

Minha querida esposa Juliana e minha filhinha Ana Paula, todo o meu amor.

Queridos e estimados pais, obrigado por serem o que são e por todo o apoio e ensinamentos que não se aprende somente nos livros e sim com exemplos de coragem, dignidade, honestidade, repeito e tantos outros.

A minha Família, em especial a tia Jurani pelo acolhimento no seu caloroso coração e a Wilma, minha sogra, obrigado pelo carinho e pela dedicação quando nós precisamos.

Aos meus amigos e em especial a Livia, Luís Reis e Viviane pelo período de companherismo, meus sinceros agradecimentos.

A FIMES que me proporciona tantos aprendizados práticos. A todos da direção, funcionários e aos alunos pelo respeito e compreensão nesta fase.

Aos demais professores da UFU, os quais eu tive a honra de ser aluno, o meu obrigado.

Sumário

Lista de Figuras	xi
Lista de Tabelas	xiii
Acrônimos	xiv
1 Introdução	1
1.1 Escopo do Trabalho	2
1.2 Estrutura da Dissertação	3
2 Descrição da Arquitetura DiffServ	4
2.1 Qualidade de Serviço - QoS	4
2.1.1 Apresentação dos Principais Parâmetros de QoS	6
2.1.2 Arquiteturas que Oferecem Suporte à QoS na Internet	9
2.2 Arquitetura de Serviços Diferenciados - DiffServ	11
2.2.1 Classificação e Condicionamento de Tráfego	11
2.2.2 Serviços Padronizados no Tratamento de Tráfego	12
2.2.3 Comparação DiffServ <i>versus</i> IntServ	16
2.3 Trabalhos Correlatos à Provisão de QoS com DiffServ	17
2.4 Considerações Finais	19
3 Descrição da Arquitetura Proposta	20
3.1 Aspectos Gerais da Arquitetura Proposta	20
3.2 Especificação da Arquitetura Proposta	22
3.2.1 Componente BB	22
3.2.2 Componente IQoS	29
3.2.3 Componente TCD	33
3.2.4 Componente Ambiente de Gerenciamento	36
3.3 Interações entre os Componentes da Arquitetura Proposta	38
3.3.1 Listagem dos Componentes TCDs Disponíveis no Domínio DS	38
3.3.2 Criação de uma Classe de Serviço DS	39
3.3.3 Especificação e Requisição de uma Banda de Transmissão	40
3.3.4 Enquadramento de um Fluxo em uma Classe de Serviço DS	41
3.3.5 Atualização do Uso de uma Classe de Serviço DS	43

4	Aspectos de Projeto e Tecnologias	44
4.1	Arquitetura CORBA	44
4.1.1	IDL - <i>Interface Definition Language</i>	46
4.1.2	Aspectos de Interoperabilidade	46
4.1.3	Java e CORBA	46
4.2	Tecnologia Web	47
4.2.1	<i>Applets</i> Java	47
4.3	Controle de Tráfego com IPROUTE2	47
4.4	Controle de Tráfego no <i>Linux</i>	49
4.4.1	Tratamento dos Pacotes	50
4.4.2	Mecanismos de Modelagem com Tokens & Buckets	51
4.4.3	Disciplinas de Fila Utilizadas	52
4.4.4	Filtros e Classificadores	58
4.5	Gerando Tráfego com o RUDE e CRUDE	59
4.6	Avaliação dos Parâmetros de QoS com TCPDump	61
4.7	Considerações Finais	61
5	Descrição dos Experimentos e Resultados	62
5.1	Avaliação das Ferramentas	63
5.2	Resposta DiffServ no Suporte à QoS	66
5.2.1	Comportamento dos Fluxos sem o DiffServ	66
5.2.2	Comportamento dos Fluxos com o DiffServ	67
5.3	Resposta da Arquitetura no Suporte à QoS	68
5.3.1	Associação de um Fluxo à uma Classe AF	70
5.3.2	Associações Conflitantes de Fluxos à uma Classe AF	71
5.3.3	Associação de Dois Fluxos à uma mesma Classe AF	73
5.3.4	Monitoração de uma Classe AF	74
6	Conclusões e Trabalhos Futuros	79
6.1	Trabalhos Futuros	80
	Referências Bibliográficas	81

Lista de Figuras

2.1	Visão do Usuário sobre QoS.	5
2.2	Pontos de Vista de um Serviço para Diferentes Usuários.	7
2.3	Parâmetros de QoS de Áudio e Vídeo.	7
2.4	Abstração dos Parâmetros de QoS em Diferentes Camadas.	8
2.5	Detecção do Estado de Congestionamento.	9
2.6	Localização do Campo DSFIELD no Cabeçalho IP.	10
2.7	Componentes da Arquitetura de Serviços Integrados (IntServ).	11
2.8	Arquitetura de Serviços Diferenciados (DiffServ).	12
2.9	Componentes da Arquitetura DiffServ.	13
2.10	Prioridade do Fluxo EF sobre o BE.	14
2.11	Filas Distintas para Suporte ao Serviço AF.	14
2.12	Arquitetura de um BB Multi-Camada.	17
2.13	Distribuição dos Elementos da Arquitetura Proposta por Rossano [29].	18
2.14	Modelo do Framework Proposto por Rocha [15].	18
3.1	Elementos da Arquitetura Proposta.	21
3.2	Propriedades e Métodos do BB.	23
3.3	Elementos que Compõem o Componente BB.	25
3.4	Diagrama da Classe classRoot.	26
3.5	Diagrama da Classe classClass.	28
3.6	Diagrama da Classe classFilter.	29
3.7	Propriedades e Métodos do IQoS.	30
3.8	Elementos que Compõem o Componente IQoS.	30
3.9	Diagrama da Classe classAVStream.	31
3.10	Diagrama da Classe classAudioFlow.	32
3.11	Diagrama da Classe classVideoFlow.	33
3.12	Propriedades e Métodos da Classe classVideoResolution.	33
3.13	Diagrama de Atividades do Objeto IQoSTest.	34
3.14	Propriedades e Métodos da Classe classTCD.	35
3.15	Elementos que Compõem o Componente TCD.	35
3.16	Propriedades e Métodos da Classe classUtils.	36
3.17	Propriedades e Métodos da Classe FrmAmbGer.	37
3.18	Listagem dos TCDs Disponíveis no Domínio DS.	38
3.19	Criação de uma Classe DiffServ.	39

3.20	Especificação e Requisição de uma Banda de Transmissão.	40
3.21	Enquadramento de um Fluxo em uma Classe de Serviço DS.	42
3.22	Atualização do Uso de uma Classe de Serviço DS.	43
4.1	Componentes da Arquitetura CORBA.	44
4.2	Localização do Controle de Tráfego no <i>Linux</i>	49
4.3	Disciplina de Fila FIFO.	49
4.4	Canal de Comunicação com Dois Fluxos Distintos.	50
4.5	Disciplina de Fila Simples sem a Utilização de Classes.	50
4.6	Disciplina de Fila com a Utilização de Classes.	51
4.7	Técnica Utilizada pela Fila HTB para Regular um Fluxo.	51
4.8	Disciplina de Fila DSMARK.	52
4.9	Marcação através da Disciplina de Fila DSMARK.	53
4.10	Disciplina de Fila GRED.	55
4.11	Modelo Hierárquico de Classes HTB.	56
4.12	Divisão de Banda de Transmissão através da Disciplina HTB.	57
4.13	Cópia do Campo DSFIELD para a Variável <code>tc_index</code>	58
5.1	Localização dos Componentes no Ambiente de Teste.	62
5.2	Avaliação da Geração de Fluxo através da Ferramenta RUDE.	66
5.3	Comportamento da Arquitetura DiffServ para Fluxos das Classes BE e AF.	67
5.4	Comportamento da Arquitetura DiffServ para Dois Fluxos da Classe AF.	68
5.5	Listagem das Filas DSMARK no Ambiente de Gerenciamento.	69
5.6	Listagem dos <i>Daemons</i> TCDs no Ambiente de Gerenciamento.	70
5.7	Comportamento de Dois Fluxos BE e um Fluxo AF.	71
5.8	Requisição uma Banda não Suportada pela Rede por uma Aplicação.	72
5.9	Contabilização da Banda de Transmissão Associada à uma Classe DS	73
5.10	Dinâmica de Dois Fluxos AF e um BE.	75
5.11	Localização dos Componentes em um Ambiente de Teste com mais Subredes.	75
5.12	Monitoração de uma Classe AF.	76

Lista de Tabelas

2.1	Associação do Campo DSFIELD com a Classes AF.	15
2.2	Associação do Campo DSFIELD com a Classe BE.	15
2.3	Associação do Campo DSFIELD com a Classe EF.	15
4.1	Convenções do Pacote IPROUTE2.	48
4.2	Parâmetros do Filtro U32.	58
5.1	Características das Máquinas Utilizadas nos Experimentos.	63
5.2	Avaliação da Banda de Transmissão no Cenário de Teste.	64
5.3	Avaliação da Banda de Transmissão e Perdas de Pacotes com a Ferramenta RUDE.	65

Acrônimos

AF	<i>Assured Forward</i>
AMW	<i>Application Middleware</i>
BA	<i>Behaviour Agregate</i>
BB	<i>Bandwidth Broker</i>
BE	<i>Best Effort</i>
CBQ	<i>Class Based Queue</i>
CIR	<i>Committed Information Rate</i>
CORBA	<i>Common Object Request Broker Architecture</i>
CRUDE	<i>Collector Real Time UDP Data Emitter</i>
DDI	<i>Dynamic Invocation Interface</i>
DiffServ	<i>Differentiated Services</i>
DS	DiffServ
DSCP	DiffServ <i>Code Point</i>
DSFIELD	DiffServ <i>Field</i>
DSI	<i>Dynamic Skeleton Interface</i>
ECN	<i>Explicit Congestion Network</i>
EF	<i>Expedited Forward</i>
FCFS	<i>First Come First Serve</i>
FIFO	<i>First In First Out</i>
FTP	<i>File Transfer Protocol</i>
GIOP	<i>Generic Inter-ORB Protocol</i>

GRED	<i>Generalized Random Early Detection</i>
HTB	<i>Hierarchical Token Bucket</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>HyperText Transfer Protocol</i>
ICMP	<i>Internet Control Management Protocol</i>
IDL	<i>Interface Definition Language</i>
IDL	<i>Interface Description Language</i>
IETF	<i>Internet Engineering Task Force</i>
IIOP	<i>Internet Inter-ORB Protocol</i>
IntServ	<i>Integrated Services</i>
IP	<i>Internet Protocol</i>
IPROUTE2	<i>Internet Protocol Route 2</i>
IQoS	<i>Intercept QoS</i>
ISC	<i>Internet Systems Consortium</i>
ISO	<i>International Standards Organization</i>
MF	<i>Multifield Classifier</i>
OMA	<i>Object Management Group</i>
ORB	<i>Object Request Broker</i>
OSI	<i>Open Systems Interconnection</i>
PC	<i>Personal Computer</i>
PDA	<i>Personal Digital Assistant</i>
PHB	<i>Per Hop Behavior</i>
QDISC	<i>Queue Discipline</i>
QoS	<i>Quality of Service</i>
RCA	<i>Resource Control Agent</i>
RCL	<i>Resource Control Layer</i>

RCP	<i>Resource Control Points</i>
RED	<i>Random Early Detection</i>
RSVP	<i>Reservation Protocol</i>
RUDE	<i>Real Time UDP Data Emitter</i>
SLA	<i>Service Level Agreement</i>
TBF	<i>Token Bucket Filter</i>
TCP	<i>Transmission Control Protocol</i>
ToS	<i>Type of Service</i>
UDP	<i>User Datagram Protocol</i>
UML	<i>Unified Modeling Language</i>
URL	<i>Uniform Resource Locator</i>
VQ	<i>Vitual Queues</i>
WWW	<i>World Wide Web</i>

Capítulo 1

Introdução

No início, a *Internet* era uma pequena rede de pesquisa utilizada por poucos usuários, onde a transferência de dados se dava através do compartilhamento de informações que indicavam as possíveis rotas a serem utilizadas. Atualmente, a *Internet* constitui o ponto de referência no que diz respeito à transmissão de dados. Segundo dados da ISC (*Internet Systems Consortium*) a *Internet* cresce vertiginosamente e de forma constante. A estimativa mínima da quantidade de computadores aproxima-se dos 285 milhões segundo o próprio órgão de pesquisa (ISC). Além do aumento de computadores e de outros equipamentos, como os celulares conectados, as aplicações, principalmente as multimídia, tais como vídeo conferência, realidade virtual, vídeo sob demanda e outras, esta rede tem sido utilizada por um número crescente de usuários o que contribui para aumentar ainda mais o volume de tráfego.

Os principais protocolos da Arquitetura *Internet* são o TCP (*Transmission Control Protocol*) e o IP (*Internet Protocol*), que estão localizados respectivamente na camada quatro e na três segundo o Modelo OSI da ISO. O TCP é responsável pela confiabilidade e o IP é responsável pelo roteamento. A tarefa de roteamento consiste em somente conferir o endereço de destino do pacote IP em uma tabela de roteamento para determinar o próximo nó. Se a fila para o próximo nó for longa, o pacote pode ser atrasado e se ela estiver cheia ou indisponível é permitido ao roteador IP descartar o pacote. O resultado é que o IP provê um serviço do tipo melhor esforço que está sujeito a atrasos imprevisíveis e a perdas de dados [1]. A simplicidade do IP vem da idéia criada logo no surgimento do IETF (*Internet Engineering Task Force*), “*IP over everything*” ou IP sobre tudo ou qualquer coisa. Esta frase caracterizava a capacidade do protocolo IP de operar praticamente sobre qualquer meio de transmissão e comunicar-se através de qualquer plataforma de sistema. Com o alto crescimento da *Internet* nos últimos anos, o surgimento de um grande número de novas aplicações, a convergência de outras redes, telefone, rádio e televisão tendo a *Internet* como meio de comunicação, “*Everything Over IP*” ou tudo sobre IP é certamente a frase adequada para os dias atuais [2].

A convergência dos mercados de entretenimento, de telefonia fixa e móvel, e da rede de dados (*Internet*) torna evidente a necessidade de maior banda de transmissão e de garantias no oferecimento de serviços. Entretanto, o máximo que se consegue na *Internet* utilizando o TCP é garantir a entrega sem perdas, duplicações e erros. Para algumas aplicações como FTP (*File Transfer Protocol*) e Telnet esta abordagem é válida, mas para muitas outras pouco resolve, pois elas necessitam de serviços não contemplados pela Arquitetura *Internet*.

As aplicações avançadas são mais exigentes que as convencionais e o que esperam da rede depende funcionalidades oferecidas. Por exemplo, aplicações de vídeo interativas exigem da rede um limite máximo para o atraso fim a fim e baixa perda de pacotes, enquanto que as aplicações de vídeo não interativas toleram maior variação do atraso. Algumas das aplicações avançadas atualmente em testes são: a telefonia sobre IP, a transmissão de áudio e vídeo de alta qualidade, videoconferência, tele-medicina, tele-diagnóstico e etc. [3].

Algumas aplicações dentro desse grupo agregadas a outros serviços (p.ex., laboratório virtual) necessitam de certas condições para um funcionamento adequado, posto que são altamente sensíveis ao estado da rede. Nessa ótica, QoS pode ser interpretada como uma capacidade da rede para que as aplicações possam cumprir a prestação do serviço segundo um contrato estabelecido. Várias propostas vêm sendo formuladas pelo IETF para prover QoS na *Internet*, dentre elas, o IntServ/RSVP (*Integrated Services/Reservation Protocol*) [4, 5], MPLS (*Multi Protocol Label Switching*) [6] e o DiffServ (*Differentiated Services*) [7] são algumas abordagens.

1.1 Escopo do Trabalho

A IntServ está apoiada em dois pilares: reserva de recursos e controle de admissão, ou seja, antes da transmissão dos pacotes se iniciar, a aplicação faz a solicitação dos serviços, o caminho é configurado e os recursos previamente alocados. A sinalização que é feita pelo protocolo RSVP, requer grandes mudanças na infra-estrutura da *Internet*, além de exigir um grande volume de sinalização e informações acerca do fluxo nos nós do trajeto. Por outro, a Arquitetura DiffServ utilizando-se do campo ToS (*Type of Service*) procura definir um mecanismo simples para tratar os pacotes com diferentes prioridades refletidas nos *bits* do próprio campo.

Nesta abordagem não há a necessidade de um estado por fluxo nem de sinalização, o que aumenta a sua escalabilidade [8]. Esta arquitetura também se destaca pela sua simplicidade. Em sua concepção, fluxos que possuem as mesmas necessidades são agregados em uma única classe. Já os parâmetros de QoS são associados às classes de prioridades diferentes. Assim, não há necessidade de se guardar informações de cada fluxo, o que representa uma vantagem em relação à escalabilidade das demais.

Entretanto, na Arquitetura DiffServ algumas questões ainda não foram resolvidas e constituem fomento à pesquisa. Dentre elas, a não especificação de como o SLA (*Service Level Agreement*) é estabelecido entre o usuário e o domínio DiffServ, como também não se especifica nenhum controle de admissão, ou seja, fluxos podem ser associados a uma classe que não mais os suportem. Por fim, pela diversidade de fluxos e parâmetros de QoS associados, a classificação dinâmica dos mesmos constitui um atrativo.

Neste trabalho propõe-se a especificação de uma arquitetura que ofereça classificação dinâmica de fluxos, reclassificação devido a alterações da banda de transmissão contratada e adaptação do fluxo de mídia, caso não seja possível a reclassificação automática face a ausência de recursos. Outro aspecto também considerado, é o mapeamento dos parâmetros de QoS da aplicação para os parâmetros da camada da rede. Para tanto, a arquitetura proposta está baseada em um *Broker* [9], ou seja, interceptador de requisições de QoS.

1.2 Estrutura da Dissertação

Este trabalho está organizado em seis capítulos, enumerados e descritos a seguir. A sequência apresentada decorre da ordem em que os temas foram desenvolvidos:

1. Introdução
2. Descrição da Arquitetura DiffServ
3. Descrição da Arquitetura Proposta
4. Aspectos de Projeto e Tecnologias
5. Descrição dos Experimentos e Resultados
6. Conclusões e Trabalhos Futuros

No Capítulo 1, apresentou-se uma visão geral sobre o funcionamento da *Internet* relacionado ao aspecto da entrega de pacotes, demonstrando a fragilidade de seu principal serviço, o melhor esforço, para as aplicações multimídias e as necessidades destas aplicações quanto aos requisitos da rede. Por fim, apresentou-se qual é o escopo e objetivos deste trabalho.

O Capítulo 2 apresenta com maiores detalhes, definições e conceitos relacionados com Qualidade de Serviço. Inicia-se com uma introdução sobre as abordagens de QoS existentes e as arquitetura propostas pela IETF. O enfoque concentra-se na Arquitetura DiffServ que é a base para este trabalho.

No Capítulo 3 o foco é a modelagem dos componentes adicionados a Arquitetura DiffServ. O processo de modelagem descreve as propriedades, eventos e os relacionamentos entre eles.

O Capítulo 4 descreve as tecnologias adotadas para a implementação da proposta. Vários pontos foram estudados, desses destacamos: sistema operacional com suporte ao controle de tráfego; ferramentas configuráveis para gerar tráfego na rede e ferramentas para análise das propriedades relacionadas aos fluxos gerados.

No Capítulo 5 apresenta-se os testes e para cada teste descreve-se em detalhe a forma como foi realizado bem como o resultado e análise obtido. O objetivo é analisar o funcionamento dos componentes e avaliar se este condiz com as propostas apresentadas no trabalho.

Finalmente, o Capítulo 6 apresenta as conclusões finais do trabalho, as principais contribuições e as perspectivas para trabalhos futuros.

Capítulo 2

Descrição da Arquitetura DiffServ

Este capítulo está dividido em duas grandes seções, a primeira descreve o tema Qualidade de Serviço, as suas várias abordagens e define qual o sentido de QoS neste trabalho e por fim, apresenta os vários parâmetros de QoS associados a camada de aplicação e a camada de rede. Na segunda seção, apresenta-se as duas principais abordagens de provisão de QoS: IntServ e o DiffServ. A Arquitetura DiffServ é detalhada em vários aspectos, é descrito os comportamentos dados aos fluxos (PHB) e os seus componentes. No final de capítulo é mostrado as deficiências e é sugerido as nossas propostas para agregar funcionalidades para suprir tais deficiências.

2.1 Qualidade de Serviço - QoS

Não existe um consenso para definir Qualidade de Serviço. Em 1999 a CISCO em um de seus relatórios [10], diz que QoS possui conotações mais específicas para algumas tecnologias de rede, tal como ATM, mas o termo é agora mais amplamente usado. Fala-se de QoS nos sistemas de armazenamento em disco, nos sistemas operacionais e entre outros. Para a área de redes, refere-se à habilidade que uma rede tem de prover melhor serviço para um determinado tráfego em tecnologias como: IP, Frame-Relay, ATM, Ethernet e outras.

No trabalho de Ferguson e Huston [11], a Qualidade de Serviço é a capacidade da rede prover um encaminhamento de dados de forma consistente e previsível. Em outras palavras, é a capacidade de satisfazer às necessidades das aplicações. Qualidade de Serviço é definida como a habilidade de um elemento da rede, seja uma aplicação, *host*, roteador, ou outro dispositivo de ter algum nível de garantia de suas exigências.

Para Teitelbaum [12] QoS é um termo que frequentemente possui duplo significado: refere-se ao desempenho de uma rede em relação às necessidades de uma aplicação e ao conjunto de tecnologias que permitem a rede satisfazer estas garantias de desempenho.

No contexto deste trabalho, Qualidade de Serviço é entendida como a capacidade da rede, através dos mecanismos de reserva de largura de banda e priorização de tráfego, em fornecer garantias de que determinados fluxos de tráfego terão tratamento diferenciado.

Habilitar QoS requer a cooperação de todos os níveis da rede, desde as camadas superiores até as inferiores, bem como, de todos os elementos fim a fim. Qualquer garantia de QoS será no máximo tão boa quanto o elo mais fraco na "cadeia" entre origem e destino (Fig. 2.1) [13].

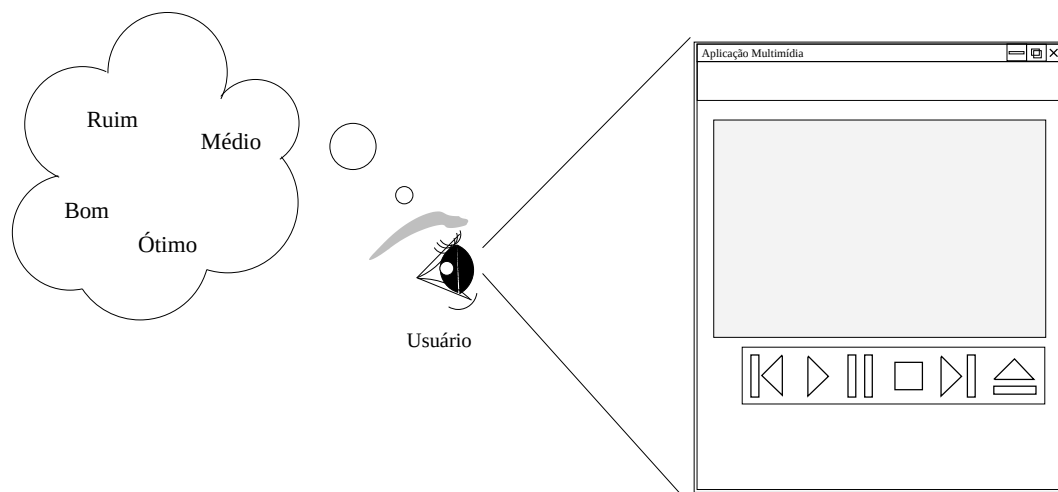


Fig. 2.1: Visão do Usuário sobre QoS.

No contexto da QoS a disponibilidade de largura de banda é, entre os demais parâmetros de rede, o mais requisitado [14]. Mecanismos de QoS administram a largura de banda de acordo com as exigências da aplicação e parâmetros de configuração e administração da rede e, desta forma, não podem prover garantias nos casos onde compartilhamento estiver envolvido. Conseqüentemente, QoS com um nível de serviço garantido requer alocação de recursos a fluxos de dados individuais. A largura de banda alocada a uma aplicação em uma reserva de recurso não estará disponível para uso em aplicações do tipo melhor-esforço. Uma vez que a largura de banda é um recurso finito, a prioridade dos projetistas de QoS tem sido assegurar que o tráfego de melhor esforço não pereça após as reservas serem efetivadas. Habilitar QoS para aplicações de alta prioridade não deve impossibilitar o uso das aplicações de baixa prioridade. No pior caso, estas últimas, devem continuar funcionando mesmo tendo um serviço mais lento.

Do ponto de vista de uma infra-estrutura de rede, requisições de uma aplicação que deseja utilizar-se de QoS são vistas como uma solicitação que será traduzida em um "contrato de serviço". Solicitações são feitas através da negociação de parâmetros cujos valores devem estar bem definidos. A *Internet* não oferece nenhum suporte na provisão, no monitoramento e na adaptação de parâmetros de QoS, que para uma determinada aplicação constitui o conjunto de parâmetros com imposições ou restrições que devem ser satisfeitas para que a aplicação opere dentro dos padrões especificados [15].

Um conjunto de parâmetros da aplicação, cujos valores descrevem o serviço oferecido, é visto pela camada de rede através de seus respectivos parâmetros. Por exemplo, dada as características de uma aplicação comum: áudio 8 *bits* por amostra com taxa de amostragem em 8 KHz e vídeo em quadros de 160x120 *pixels* com 24 *bits* por *pixel* e 20 quadros por segundo, qual deve ser a largura de banda necessária para acomodar tal fluxo? Observa-se para este exemplo que uma vazão de 9,3 Mbps é suficiente para atender a aplicação. Através deste exemplo, mostra-se como a tradução dos parâmetros da aplicação para os da rede pode se dar. Nem todos os parâmetros da camada da aplicação podem ser traduzidos somente para o parâmetro largura de banda, não obstante, trata-se apenas de parâmetros de áudio e vídeo que podem ser traduzidos para o parâmetro largura de banda.

$$banda_video = 160 \cdot 120 \cdot 24 = 9.216.000bps$$

$$banda_audio = 8 \cdot 8000 = 64.000bps$$

$$banda_total = \frac{(banda_video + banda_audio)}{10^6} = 9.3Mbps$$

Essencialmente existem dois mecanismos básicos disponíveis para prover QoS em redes IP:

- Reserva de Recursos: os recursos da rede são divididos de acordo com os requisitos de QoS da aplicação, e sujeitos à política de administração de largura de banda. O RSVP (*Resource ReSerVation Protocol*), por exemplo, fornece os mecanismos para implementação de Serviços Integrados [4] baseado na reserva de recursos.
- Priorização: o tráfego da rede é classificado e os recursos de rede são divididos de acordo com critérios de políticas de administração de largura de banda. Para habilitar QoS os mecanismos de classificação asseguram tratamento preferencial a aplicações identificadas como tendo exigências quanto aos seus requisitos. A arquitetura de Serviços Diferenciados [7, 16] provê este tipo de serviço.

Esses dois mecanismos não são mutuamente exclusivos. Ao contrário, eles são complementares. Ou seja, eles são projetados para uso em conjunto, de forma a acomodar diferentes exigências operacionais em diferentes contextos de redes.

2.1.1 Apresentação dos Principais Parâmetros de QoS

A camada de aplicação e a camada de rede possuem seus próprios parâmetros de QoS para expressar o seu requisito quanto a um serviço. Esta seção destaca e descreve os principais parâmetros da camada da aplicação e da camada de rede da Arquitetura TCP/IP.

Os usuários expressam seus desejos em termos associados à própria linguagem humana para referenciar um tipo de serviço esperado. É comum a utilização de parâmetros como: ruim, médio, bom e ótimo. Esses parâmetros são extremamente subjetivos, dependendo do usuário e de seu contato com a computação, parâmetros com valores iguais podem ter significados diferentes para usuários diferentes. Por exemplo (Fig. 2.2), e bem provável que uma pessoa que nunca teve contato com equipamentos eletrônicos, ao ver um vídeo de uma *resolução pequena*, sem cor (apenas preto e branco) fique admirado com a apresentação e diga, surpreso, “Que ótimo!!!”. Já uma outra pessoa habituada com o mundo digital ao ver o mesmo vídeo diga: “Que péssimo!!!”. Podemos observar que esses parâmetros são individuais e dependem das experiências de cada usuário quanto a um serviço. Ainda neste exemplo o termo *pequena resolução* também pode ter diferentes nuances para cada leitor.

Após o usuário selecionar os parâmetros de QoS que uma aplicação disponibiliza, a aplicação os converte para parâmetros que de fato, estão relacionados ao tipo de mídia a ser tratada.

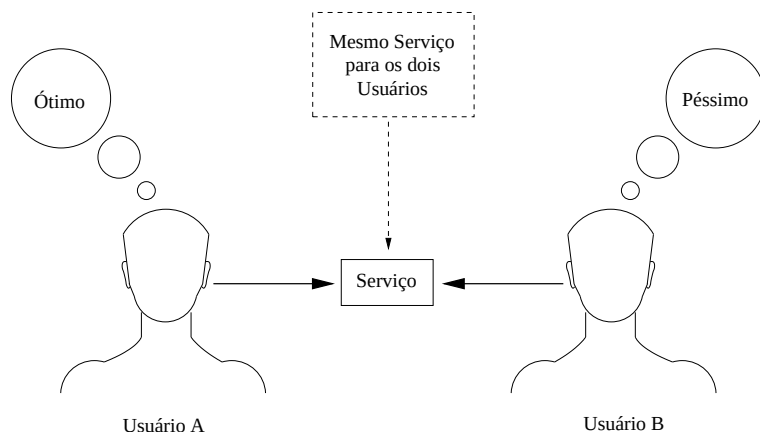


Fig. 2.2: Pontos de Vista de um Serviço para Diferentes Usuários.

Também fazem parte do conjunto de parâmetros da aplicação, valores que são utilizados para reproduzir sensações analógicas como: imagens e sons. Dependendo dos valores dos parâmetros a reprodução poderá ser mais ou menos fiel à realidade (Fig. 2.3). Segundo a especificação *Audio/Video Stream* [17] os parâmetros relacionados com áudio e vídeo são:

- `audio_sampleSize`: quantidade de *bits* para armazenar uma amostra;
- `audio_sampleRate`: taxa de obtenção de amostras;
- `audio_numChannels`: quantidade de canais;
- `audio_quantization`: tipo de modulação/codificação do áudio (0 = linear, 1 = law, 2 = A-law, 3 = GSM);
- `video_framerate`: quantidade de *frames* (quadros) por segundo;
- `video_colorDepth`: quantidade de *bits* para armazenar um ponto de uma imagem pertencente ao frame do vídeo;
- `video_resolution`: valores referentes a altura e a largura do vídeo.

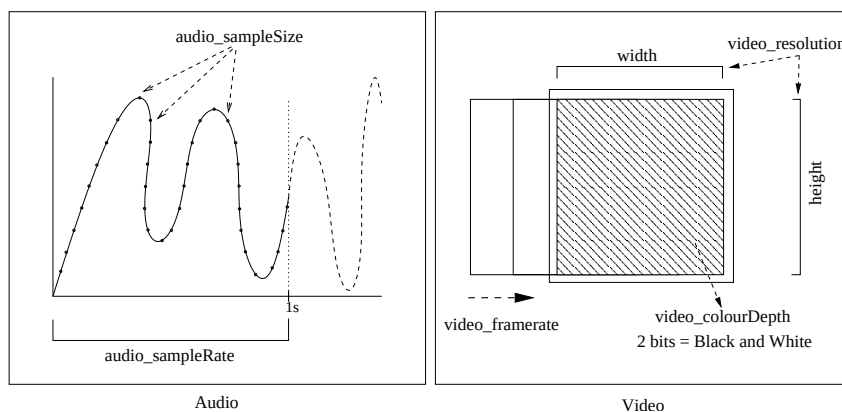


Fig. 2.3: Parâmetros de QoS de Áudio e Vídeo.

Os parâmetros considerados essenciais para redes que implementam mecanismos de QoS (Fig. 2.4), segundo a camada de rede são: a minimização do atraso fim a fim e da variação do atraso (*jitter*); a minimização da taxa de perda de pacotes; o provimento de vazão de dados; largura de banda. Esses parâmetros são detalhados nos seguintes itens:

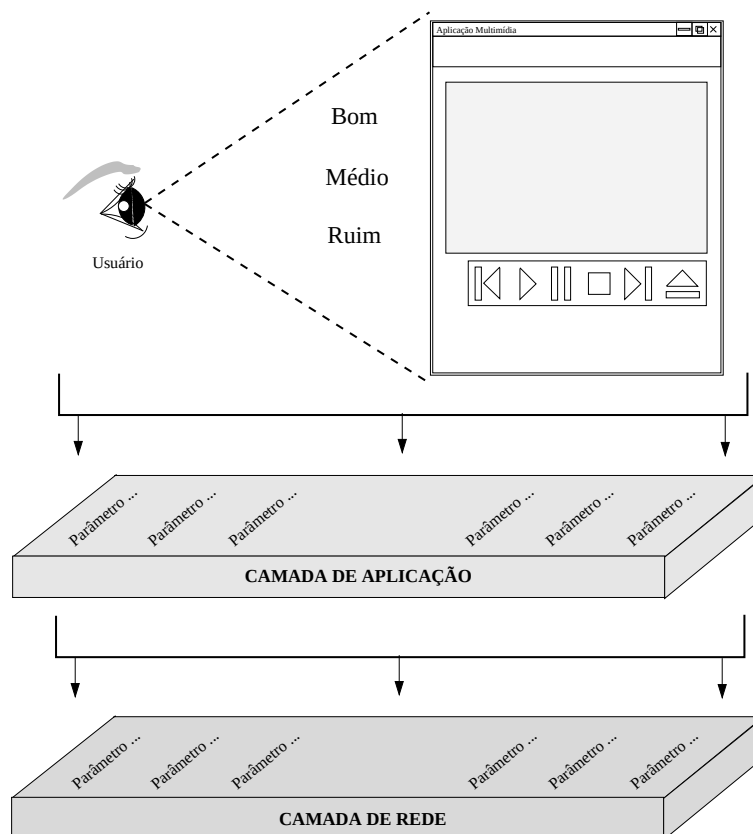


Fig. 2.4: Abstração dos Parâmetros de QoS em Diferentes Camadas.

- atraso fim a fim: é o tempo entre o envio de uma mensagem por um nó e a recepção desta mensagem pelo nó destino. Este atraso ocorre no caminho de transmissão ou em um dispositivo no caminho de transmissão. Em um roteador, o atraso é o montante de tempo entre a recepção do pacote e a sua transmissão. Este tempo é também referido como atraso de propagação;
- variação do atraso (*jitter*): em redes de pacotes, este parâmetro representa a distorção ocorrida nos tempos de chegada entre pacotes comparados aos tempos anteriores de transmissão entre pacotes. É uma distorção que acontece, por exemplo, quando fluxos de áudio ou vídeo são transmitidos em uma rede e os pacotes não chegam no seu destino dentro da ordem sucessiva ou em uma determinada cadência. Ou seja, eles variam em termos de tempo de atraso. Esta distorção é particularmente prejudicial ao tráfego multimídia, fazendo com que o sinal de áudio ou vídeo tenham uma qualidade distorcida ou fragmentada na recepção;

- taxa de perda de pacotes: a taxa perda de pacotes representa o percentual de pacotes que foram transmitidos na rede, mas não alcançaram seu destino em um determinado período de tempo (Fig. 2.5). Este parâmetro é essencial para determinar se uma rede, por exemplo, está congestionada. O congestionamento na rede é determinado quando o número de pacotes aumenta, enquanto os pacotes entregues diminuem;

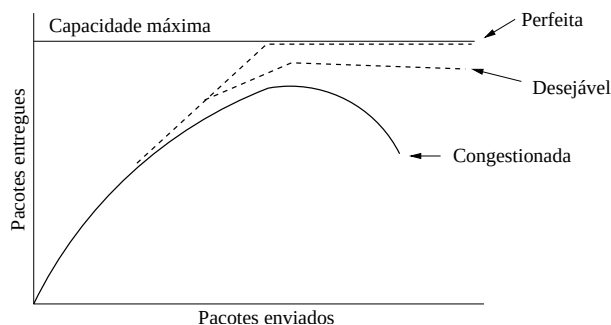


Fig. 2.5: Detecção do Estado de Congestionamento.

- valor da largura de banda: largura de banda é uma medida de capacidade de transmissão de dados, normalmente expressa em *kilobits* por segundo (Kbps) ou *megabits* por segundo (Mbps). A largura de banda indica a capacidade máxima de transmissão teórica de uma conexão. Entretanto, na medida em que a taxa de transmissão utilizada se aproxima da largura de banda teórica máxima, fatores negativos como atraso na transmissão das informações podem causar deterioração na qualidade. A largura de banda de uma rede pode ser vista como um tubo que transfere dados. Quanto maior o diâmetro do tubo, mais dados podem ser enviados através dele simultaneamente. A vazão é o montante de tráfego de dados movidos de um nó da rede para outro em um determinado período de tempo. A vazão também é expressa em Kbps ou Mbps.

2.1.2 Arquiteturas que Oferecem Suporte à QoS na Internet

Um modelo de serviço, também chamado nível de serviço, descreve um conjunto de capacidades de QoS fim a fim. A QoS fim a fim é a habilidade da rede em prover o serviço requerido por uma aplicação (p. ex., áudio sob demanda, vídeo sob demanda, teleconferência e etc..) a uma extremidade a outra da rede [18], os serviços mais abordados são: melhor esforço, serviços diferenciados (DiffServ) e serviços integrados (IntServ).

O serviço de melhor esforço é conhecido como o sem garantia de QoS e, provê simplesmente a conectividade básica sem garantias. A *Internet* é um bom exemplo de serviço de melhor esforço. Embora o serviço de melhor esforço não ofereça garantia de QoS, ele proporciona um ponto de referência quanto à rigidez da especificação e, apesar de não oferecer garantias, é adequado para uma grande gama de aplicações de rede, tais como, correio eletrônico e transferência de arquivos, pois estas adaptam-se facilmente às condições da rede.

Na abordagem dos serviços diferenciados, uma parte do tráfego é tratada de forma privilegiada sobre o restante. É oferecida manipulação mais rápida, mais largura de banda na média e menor taxa de perda. Esta é uma preferência estatística, não uma garantia rígida. Com

métodos adequados, incluindo a utilização de determinadas políticas nas extremidades da rede, a arquitetura de serviços diferenciados pode prover um tratamento adequado para uma boa gama de aplicações, incluindo aquelas de missão crítica, as que necessitam de baixo atraso, aplicações de voz sobre pacotes, e outras. Na maioria das vezes, serviços diferenciados estão associados com a classificação do tráfego. O tráfego é agrupado em um pequeno número de classes e cada classe recebe um serviço na rede.

O grupo do IETF trabalha na especificação e definição de um padrão para os serviços de rede sob o nome genérico de DiffServ. Este esforço está focado em grande parte no uso do campo ToS (*Type of Service* - Tipo de Serviço) do cabeçalho do protocolo IP versão 4, que foi nomeado para campo DSFIELD, como um mecanismo de sinalização de QoS [16]. A Fig. 2.6 ilustra o uso deste campo. Os *bits* de 0 a 5 representam o DSCP (*Differentiated Service Code Point*), os *bits* 6 e 7 não são utilizados.

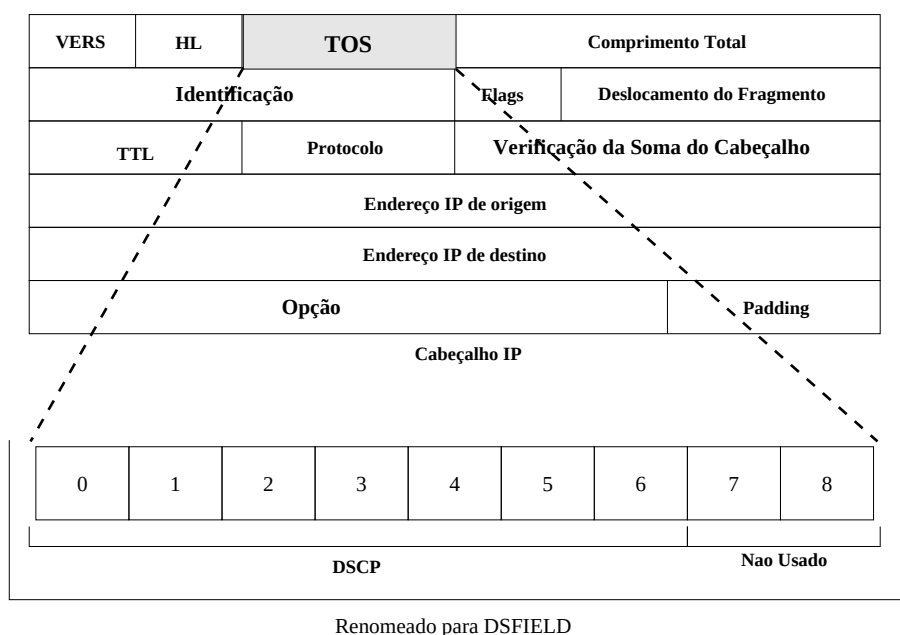


Fig. 2.6: Localização do Campo DSFIELD no Cabeçalho IP.

De acordo com a abordagem dos Serviços Integrados, a aplicação solicita que um conjunto de parâmetros possam ser garantidos pela rede antes do envio dos dados. A solicitação é feita através de uma sinalização explícita; a aplicação informa à rede seu perfil de tráfego e requisita um tipo específico de serviço capaz de satisfazer suas exigências, por exemplo, largura de banda e atraso. É esperado da aplicação que os dados sejam enviados somente após receber a confirmação da rede e que estes estejam dentro do perfil de tráfego descrito.

Tipicamente, os Serviços Integrados estão associados com o nível de classificação do tráfego e refere-se ao fluxo individual, ou seja, determinados fluxos devem ter recursos de rede reservados para alcançarem suas exigências (Fig. 2.7).

O grupo de trabalho IntServ [4, 5] do IETF desenvolveu padrões e definições específicos para os serviços definidos sob o termo genérico de Serviços Integrados. Este esforço tentou prover uma arquitetura consistente para especificar fluxos na *Internet* com exigências variadas. Os

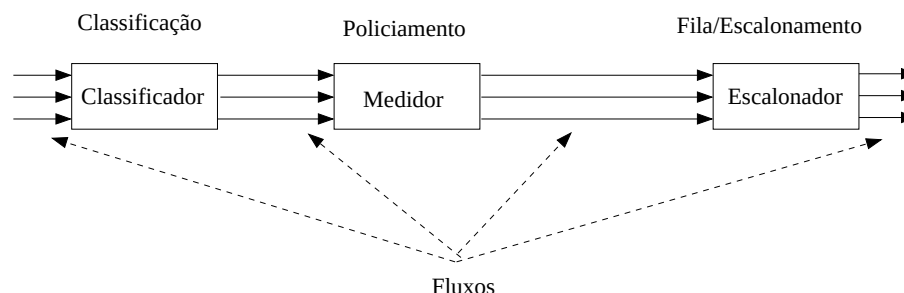


Fig. 2.7: Componentes da Arquitetura de Serviços Integrados (IntServ).

serviços primários em uso são o Serviço de Carga Controlada e o Serviço Garantido. O RSVP foi desenvolvido como um mecanismo de sinalização de QoS para prover serviços baseados nestes tipos de fluxo. Embora a técnica de reserva possa ser usada para oferecer garantias de desempenho, existe vários problemas que atrapalham sua adoção na *Internet*. Um dos principais problemas é que a maioria dos protocolos de sinalização são muito complexos e ineficazes, especialmente quando todos os casos de erros possíveis são levados em consideração. Além do mais, o fluxo de tráfego ao qual o pacote pertence precisa ser determinado em cada elemento que compõem seu caminho tornando esta arquitetura pouco escalável [19].

2.2 Arquitetura de Serviços Diferenciados - DiffServ

A Arquitetura DiffServ [20, 16, 21, 22] posiciona-se entre os extremos do melhor esforço (BE - sem priorização) e Serviços Integrados com reserva de recursos e grande sobrecarga de sinalização. Na Arquitetura DiffServ não existe alocação explícita de recursos e não é feita sinalização, tendo em vista que a prioridade do pacote é transmitida no cabeçalho IP, permitindo desta forma maior escalabilidade e baixa sobrecarga de sinalização.

O diagrama da Fig. 2.8 mostra onde os componentes-chave de uma rede DiffServ (DS) são empregados. O DiffServ define o conceito de domínio DS, que é um conjunto contíguo de nós DS que aplicam um conjunto comum de políticas sobre o tráfego que atravessa o domínio. Um domínio DS tem nós de borda e nós de interior [9] e [33]. Os nós DS de borda são responsáveis pela classificação e condicionamento do tráfego que entra no domínio DS.

Para cada fluxo entrante no domínio pelos nós de borda, a política define qual terá um serviço diferenciado, como este deverá ser marcado e como será tratado pelos nós interiores. Os nós de núcleo, examinam a marcação dos pacotes e atuam de acordo com as políticas definidas ou ao perfil de tráfego. O perfil de tráfego, conhecido como SLA é um acordo do usuário com o administrador do domínio DS na qual estabelece os parâmetros que determinarão o serviço.

2.2.1 Classificação e Condicionamento de Tráfego

A Arquitetura DiffServ (Fig. 2.9) define dois importantes componentes nos nós DS: componentes de classificação e de condicionamento de tráfego. Estas funções são mais complexas em nós DS de borda que em nós de interior, mas ambos os nós têm um classificador.

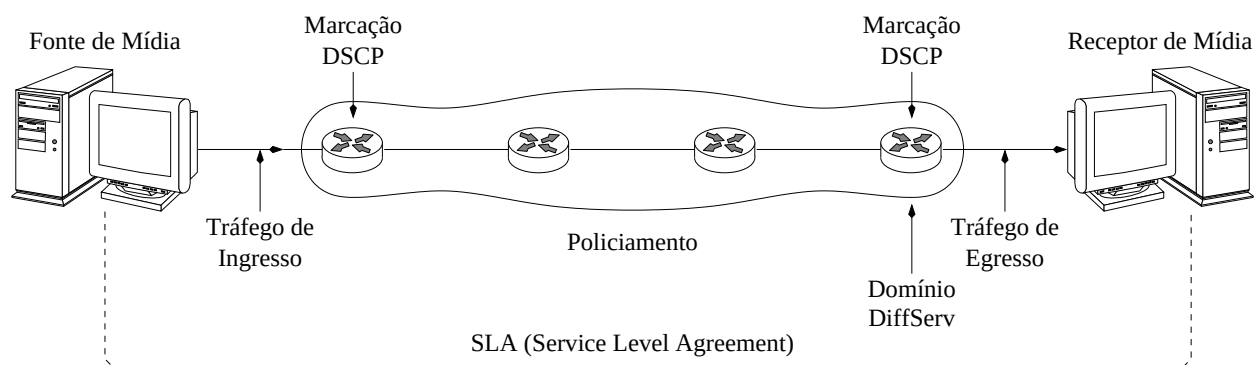


Fig. 2.8: Arquitetura de Serviços Diferenciados (DiffServ).

Existem dois tipos de classificadores: um que classifica o fluxo baseado apenas na classificação DS e outro que verifica múltiplos campos no cabeçalho IP. Estes classificadores são conhecidos como classificador de comportamento agregado (BA *Behaviour Agregate*) e classificador multi-campo (MF *Multifield Classifier*), respectivamente. A marca de classificação DS é conhecida como DSCP (DS - *Code Point*).

O classificador é o componente que divide o fluxo de entrada em um conjunto de fluxos de saída por meio de filtros que utilizam o conteúdo do cabeçalho do pacote. De acordo com a chegada do pacote o medidor verifica se o pacote está de acordo com um perfil de tráfego pré-definido. Dependendo da conformidade várias ações podem ser executadas, três tipos de ações, e a combinação delas, podem ser executadas: marcação, atraso e descarte. Escalonamento é o processo de decidir no momento da transmissão qual a fila entre um conjunto de candidatas deve ser servida, dependendo de alguma propriedade da fila e/ou do pacote. Geralmente ele é empregado no caso de contensão de recursos na saída. A ação de modelagem ou policionamento modifica o tráfego de entrada para forçar um determinado perfil de saída.

Uma fila, por sua vez, é uma área de memória necessária para executar escalonamento e/ou formatação. Condicionadores de tráfego são empregados em um determinado estágio do caminho dos dados para forçar uma determinada política. Eles podem ser implementados através da combinação de um ou mais componentes de DiffServ definidos anteriormente (classificadores, medidores, elementos de ação e filas) ou, alternativamente, através da combinação de condicionadores de tráfego existentes. Marcação é um exemplo de condicionamento de tráfego.

2.2.2 Serviços Padronizados no Tratamento de Tráfego

Como foi citado, nos nós de borda, o fluxo de tráfego é classificado e marcado. Os campos DSCP [20] são mapeados para os PHBs (*Per Hop Behaviors*) definidos na Arquitetura DiffServ. Os PHBs [22, 21] definem o comportamento de encaminhamento de um pacote em um nó DiffServ. PHBs são identificados através de um *label* de 6 *bits*, do campo TOS (Type Of Service) do cabeçalho do pacote IPv4 e o campo *Class* do cabeçalho do pacote IPv6 agora chamados de Differentiated Services Code Point (DSCP) conforme mostra a Fig. 2.6

Os seis *bits* do campo DS são usados para selecionar o PHB que o pacote terá em cada nó. Este campo é tratado como um índice em uma tabela usada para selecionar o mecanismo de

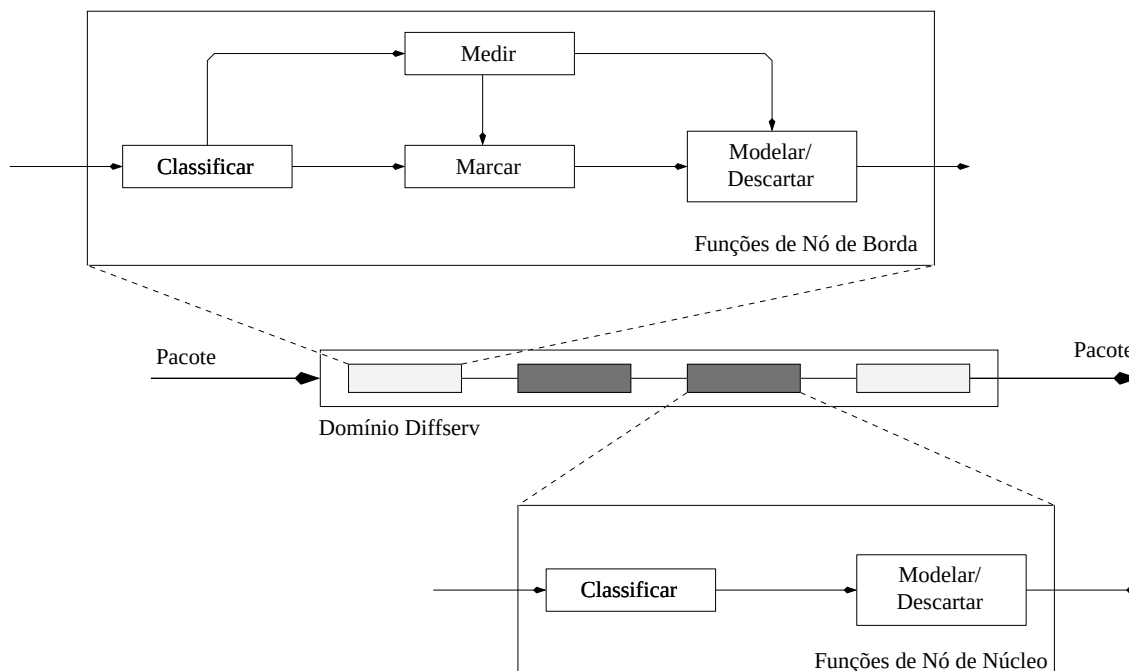


Fig. 2.9: Componentes da Arquitetura DiffServ.

manipulação de pacotes implementado em cada dispositivo. Este campo é definido como um campo não estruturado para facilitar a definição de futuros PHBs. Outros comportamentos podem especificar que, para determinados pacotes, serão dadas certas prioridades relativas a outros, em termos de vazão (*throughput*) ou de preferência para descarte, mas sem ênfase particular em atraso. PHBs são implementados utilizando mecanismos de enfileiramento.

Os PHBs são comportamentos individuais aplicados em cada roteador, por isso isoladamente não garantem QoS fim a fim. Entretanto, a interligação de roteadores com os mesmos PHBs e a limitação da taxa em que os pacotes são enviados para um PHB, possibilita o uso de PHBs para construir QoS fim a fim. Por exemplo, a concatenação de EF PHBs ao longo de uma rota preestabelecida, com um cuidadoso controle de admissão, pode prover um serviço similar ao de uma linha dedicada, que é satisfatório para voz interativa. Uma outra concatenação de PHBs pode ser utilizada para transmissão de vídeo armazenado, e assim por diante.

O PHB padrão assegura compatibilidade com o encaminhamento melhor esforço padrão em todos os roteadores, ele é documentado no RFC1812 [23]. PHBs de tráfegos com DSCP mais alto devem ter encaminhamento preferencial sobre aqueles com valor mais baixo. Por exemplo, o PHB 111000 deve ter prioridade mais alta que 000000. Dois importantes PHBs foram padronizados: o EF (*Expedited Forwarding*) que é documentado na especificação RFC2598 [21] e o AF (*Assured Forwarding*) documentado na RFC2597 [22].

O EF (Fig. 2.10), também referido como serviço *premium* ou de canal dedicado, pode ser usado para tráfego com requisitos de baixa perda, baixo atraso, baixo *jitter* e garantia de largura de banda. Estes requisitos são alcançados assegurando-se que os agregados de tráfego encontram nenhum ou pouco enfileiramento. As implementações devem prover meios de limitar o dano que o tráfego EF pode infligir sobre o outro tráfego. Os dispositivos de borda do domínio

DS devem policiar o tráfego EF para assegurar a taxa de *bits* definida. Caso os fluxos cheguem a uma taxa acima da estabelecida, o pacotes em excesso devem ser descartados.

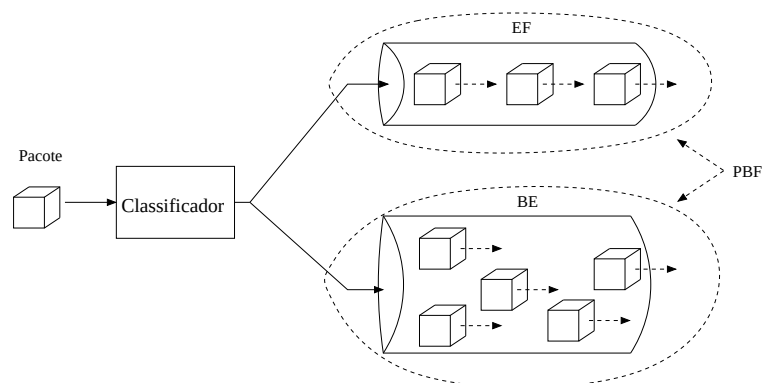


Fig. 2.10: Prioridade do Fluxo EF sobre o BE.

O AF (Fig. 2.11) tem por objetivo fornecer entrega de pacotes IP, com largura de banda assegurada, em quatro classes de transmissão, mas não oferece garantias quanto ao atraso. Cada classe tem três precedências de descartes (*Drop Precedence*), as quais são utilizadas para determinar a importância do pacote. Assim, um nó congestionado dá preferência para serem descartados, entre os pacotes de uma mesma classe, os com maiores valores de precedência.

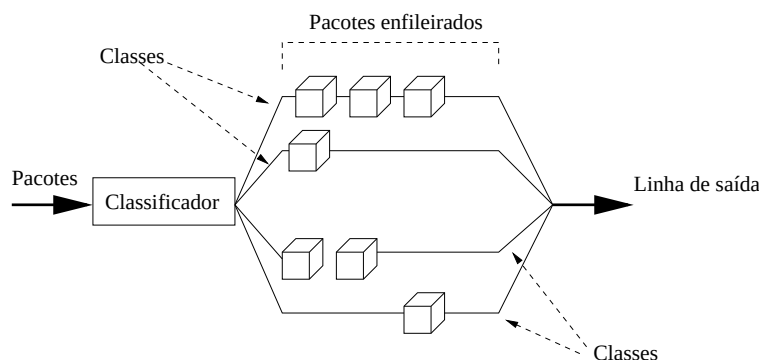


Fig. 2.11: Filas Distintas para Suporte ao Serviço AF.

Os três primeiros *bits* do DSCP identificam a classe de transmissão, 001, 010, 011 e 100 e os três últimos *bits* definem a precedência de descarte 010 para a precedência mais baixa (ou seja, último a ser descartado), 100 para precedência média e 110 para a mais alta precedência de descarte (ou seja, primeiro a ser descartado). Na Tab. 2.1, Tab. 2.2 e Tab. 2.3 são apresentados os DSCP recomendados para as classes de tráfego AF, BE e EF.

A marcação da precedência de descartes para tráfegos AF pode seguir dois enfoques: marcador com taxa simples [24] e marcador com taxa dupla [24]. Ambos usam um regulador do tipo duplo balde furado (*Dual Leaky Bucket*), sendo que no marcador simples, os dois baldes são preenchidos na mesma taxa e no marcador com taxa dupla, os baldes são preenchidos com duas taxas diferentes. Ambos os enfoques medem o tráfego e marcam o pacote, dependendo se o fluxo excede a taxa contratada (CIR - *Committed Information Rate*). Um pacote é marcado

como verde, amarelo ou vermelho dependendo em quanto o CIR foi excedido. Um pacote verde tem a menor prioridade de descarte, enquanto que um pacote vermelho, a mais alta. O principal objetivo é fornecer a marcação para algum mecanismo de descarte baseado na precedência de descartes, tal como o mecanismo RED (*Random Early Detect*) [25] .

Prioridades das Classes AF								
	Classe AF1		Classe AF2		Classe AF3		Classe AF4	
Precedência	binário	decimal	binário	decimal	binário	decimal	binário	decimal
Baixo	001010	10	010010	18	011010	26	100010	34
Médio	001100	12	010100	20	011100	28	100100	36
Alto	001110	14	010110	22	011110	30	100110	38

Tab. 2.1: Associação do Campo DSFIELD com a Classes AF.

Prioridade da Classe BE		
	Classe BE	
Precedência	binário	decimal
N/A	000000	0

Tab. 2.2: Associação do Campo DSFIELD com a Classe BE.

Prioridade da Classe EF		
	Classe EF	
Precedência	binário	decimal
N/A	101110	46

Tab. 2.3: Associação do Campo DSFIELD com a Classe EF.

Na Arquitetura DiffServ o serviço está associado à QoS do ponto de vista da aplicação, permitindo especificar às necessidades destas em termos de largura de banda, atraso, jitter e taxa de perdas, e pode ser quantitativo ou qualitativo. No primeiro caso o serviço é especificado através de um conjunto de métricas e valores de referência correspondentes, enquanto que no segundo caso somente uma definição de alto nível é fornecida. Em ambos os casos a implementação de QoS está baseada em uma determinada combinação dos componentes da Arquitetura DiffServ. Quanto aos serviços oferecidos por um domínio DS, devemos notar:

- serviços DS são todos para tráfego uni-direcional apenas;
- serviços DS são para tráfegos agregados, não fluxos individuais. Um Serviço é definido como o tratamento global de um subconjunto do tráfego do cliente dentro de um domínio aderente a DS ou fim a fim. O tráfego na rede geralmente atravessa uma concatenação de redes que podem incluir *hosts*, redes residenciais e de escritório, redes corporativas/campus e várias redes de longa distância. Redes residenciais e de escritório são

normalmente clientes de redes de campus ou corporativas, que são por sua vez clientes de redes de longa distância. Note-se que existem várias fronteiras, cliente-provedor, em que o conceito de serviço se aplica [7].

Uma política relaciona um perfil de tráfego com ações para serem efetuadas neste perfil [26]. O perfil de tráfego determina quais os requisitos (como endereços fonte e destino, portas fonte e destino, protocolo e DSCP) sob os quais um fluxo de tráfego é identificado. Uma vez classificado um pacote (identificado o seu perfil de tráfego) é aplicada à ação correspondente ao perfil identificado de acordo com sua política.

2.2.3 Comparação DiffServ *versus* IntServ

A Arquitetura Serviços Integrados/RSVP representa uma mudança fundamental na arquitetura da Internet, que é fundamentada no conceito de que todas as informações de estado relacionadas aos fluxos deveriam estar nos sistemas finais. Neste sentido, existem alguns problemas com a Arquitetura Serviços Integrados [27]:

- o montante de informações de estado aumenta proporcionalmente ao número de fluxos. Isto causa uma sobrecarga de armazenamento e processamento nos roteadores. Portanto esta arquitetura é pouco escalável;
- os requisitos nos roteadores são altos: todos os roteadores devem implementar RSVP, controle de admissão, classificação MF e escalonamento de pacotes;
- para Serviço Garantido, toda a rede deve suportar IntServ. Uma instalação gradativa de Serviço de Carga Controlada é possível;
- IntServ/RSVP não são aplicáveis em aplicações do tipo navegadores WWW, onde a duração de um fluxo típico é apenas de poucos pacotes. A sobrecarga causada pela sinalização RSVP poderia facilmente deteriorar o desempenho da rede.

Por outro lado na Arquitetura DiffServ, destacam-se as seguintes características:

- existência de apenas um número limitado de classes de serviço indicadas no campo DS. Desde que o serviço é alocado na granularidade de uma classe, o conjunto de informações de estado é proporcional apenas ao número de classes e não proporcional ao número de fluxos. Serviços Diferenciados constituem, portanto, serviços escaláveis;
- as operações de classificação, marcação, policiamento e retardo são necessárias somente nas fronteiras das redes e dependendo das políticas definidas, os fluxos de tráfego com requisitos similares são agregados. Assim, os roteadores intermediários necessitam apenas implementar a classificação BA, que é uma classificação baseada apenas no *byte* DS. Portanto, DiffServ é mais fácil de implementar e usar; e

- no modelo Serviços Diferenciados, um Serviço Assegurado (AF) pode ser fornecido por um sistema que suporta parcialmente os Serviços Diferenciados. Os Roteadores que não suportam DiffServ simplesmente ignoram os campos DS dos pacotes e fornecem serviço assegurado ou serviço de melhor esforço. Como os pacotes de serviço assegurado têm menos probabilidade de serem descartados em roteadores compatíveis com DS, o desempenho total do tráfego de Serviço Assegurado será melhor que o tráfego Melhor Esforço.

Como consequência da comparação entre os dois modelos de provimento de QoS analisados motivaram a escolha pela implementação DiffServ. Percebeu-se que estas facilidades apontadas em termos de implementação, escalabilidade e, principalmente, o fato de não ser necessário modificar as aplicações, visto que a classificação, marcação e priorização dos fluxos de tráfego podem ser feitas nos roteadores de borda, viabilizariam a implantação imediata do serviço.

2.3 Trabalhos Correlatos à Provisão de QoS com DiffServ

Várias são as abordagens e propostas de uso do DiffServ no suporte ao oferecimento de QoS. Na proposta de Politis [28], Figura 2.12, o BB (*Bandwidth Broker*) é incluído e estruturado em uma entidade lógica chamada de RCL (*Resource Control Layer*). A RCL é dividida em duas subcamadas: a primeira, composta pelo RCP (*Resource Control Point*), é responsável por gerenciar e distribuir recursos na rede. A segunda é formada por dois componentes, o RCA (*Resource Control Agent*), responsável pelo controle de admissão e o AMW (*Application Middleware*), responsável por disponibilizar uma interface a partir da qual o usuário fornece os parâmetros de QoS. A ênfase do trabalho está na modelagem do gerenciamento de QoS em um domínio DiffServ, mostrando a importância de se separar esses elementos.

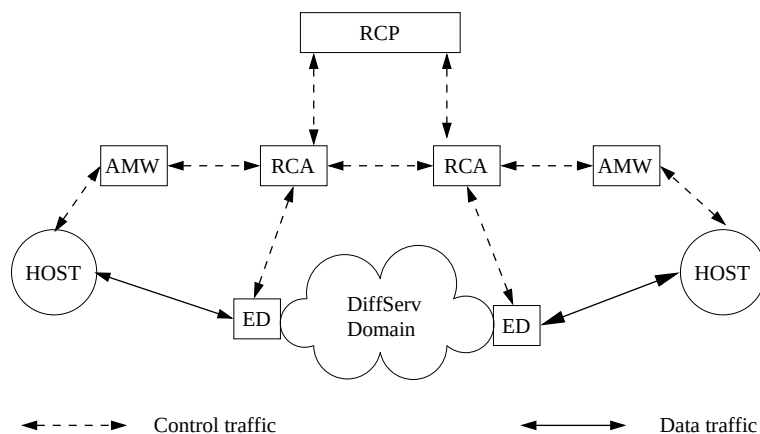


Fig. 2.12: Arquitetura de um BB Multi-Camada.

Entretanto, para aplicações telemáticas, enquadradas como aplicações de tempo real, a negociação dos parâmetros de QoS não é conhecida pelo usuário. Para contornar este problema, Rossano [29], Figura 2.13, emprega além de um BB, um interceptador de QoS (IQoS). O IQoS é responsável pela negociação e mapeamento dos parâmetros de QoS da aplicação para a rede.

Adicionalmente, uma implementação sobre plataforma Linux foi desenvolvida e testes com aplicações contemplando fluxos de áudio/vídeo confirmaram o mapeamento automático.

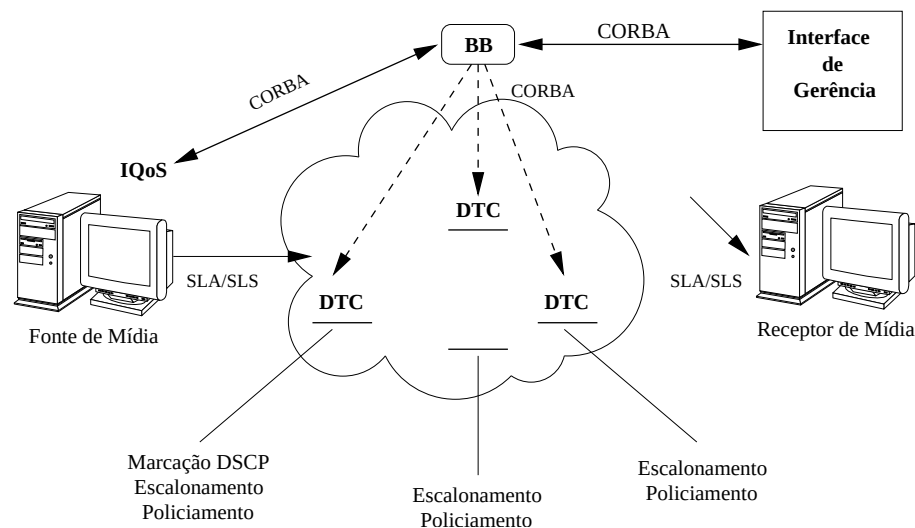


Fig. 2.13: Distribuição dos Elementos da Arquitetura Proposta por Rossano [29].

Na linha de especificação, Rocha [15] apresenta um *framework* genérico, Figura 2.14, para provisão de QoS em Redes IP. Para tanto, propõe duas camadas: a de baixo nível utiliza o RSVP (*ReSerVation Protocol*), de modo a oferecer uma interface para protocolos como o TCP/UDP, bem como o gerenciamento de dispositivos de rede; e a camada superior tem por objetivo facilitar a especificação e monitoramento dos parâmetros de QoS pelos programadores da aplicação. Dentre as questões propostas no *framework*, destaca-se a preocupação na disponibilidade de uma interface de alto nível capaz de acomodar aspectos da especificação, negociação e adaptação dos parâmetros de QoS.

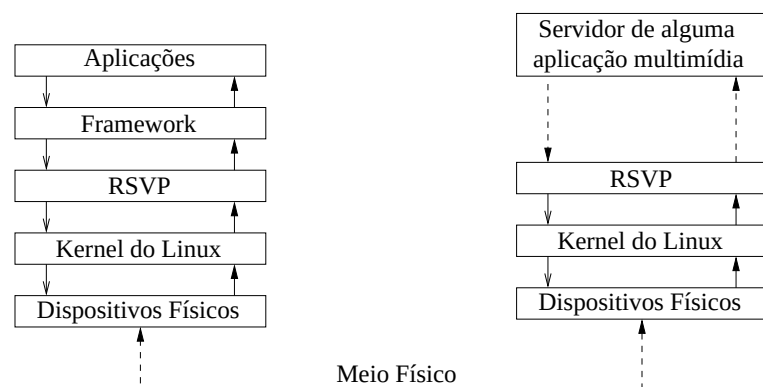


Fig. 2.14: Modelo do Framework Proposto por Rocha [15].

Na linha de trabalhos práticos, Kuznetsov [30] apresenta um estudo que utiliza a plataforma Linux e o Pacote IPROUTE2 [30] com o intuito de avaliar o desempenho das ferramentas e

do comportamento das disciplinas de fila suportadas pelo *kernel*. Adicionalmente, constituiu objeto de avaliação a classificação e policiamento dos roteadores de borda e de núcleo do domínio DiffServ. Dentre os resultados apresentados, destaca-se a importância na escolha da disciplina de policiamento de uma determinada classe. Dentre as disciplinas avaliadas, a TBF (*Token Bucket Filter*) mostrou-se mais adequada que a CBQ (*Class Based Queue*), quando não se permite o policiamento fora da banda previamente definida.

A Arquitetura DiffServ apresenta algumas desvantagens, os serviços diferenciados dão segurança ao desempenho das aplicações somente em termos relativos, o esquema de prioridade relativa garante que uma aplicação gerando tráfego de determinada prioridade terá melhor desempenho que outra gerando tráfego de menor prioridade. Dependendo da carga da rede, ambas as aplicações podem ter um desempenho muito aquém das suas reais necessidades.

As deficiências que merecem destaque na Arquitetura DiffServ, são: falta de um controle de admissão, falta de um mecanismo para controle de recursos e de monitoramento. Sua especificação se baseia em não oferecer uma garantia absoluta dos parâmetros de QoS, mas acomodar vários fluxos de mídia numa mesma classe. Cada classe possui uma largura de banda associada, quando o usuário deseja utilizar um serviço ele informa os valores dos parâmetros que atendem as suas necessidades. A classe escolhida será a que melhor se ajuste a elas.

Em razão da dificuldade em prever um tráfego na rede, nem sempre será possível acomodar fluxos da aplicação de um usuário dentro de uma classe que atenda aos parâmetros de contrato. A criação das classes utilizadas para acomodar um conjunto de fluxos, possuem um valor numérico inteiro, que corresponde à largura de banda reservada a esta classe. Várias classes podem ser criadas, desde que a soma das bandas não ultrapasse a banda total da rede. Ao associar um fluxo a uma classe, não é suficiente apenas verificar se a classe possui recursos, neste caso banda, para atender o requisito da aplicação, pois se vários fluxos forem associados a uma mesma classe e se a quantidade de fluxo gerado for maior que a vazão dessa classe os fluxos receberão tratamento de melhor esforço. Adicionalmente, definir a quantidade de classes e suas respectivas vazões constitui outra questão de projeto. Nos trabalhos anteriormente discutidos, nenhuma consideração é feita acerca do número de classes face a dinâmica da rede.

Um outro aspecto a considerar é que o usuário conhece as exigências quanto à qualidade que espera no oferecimento do serviço, mas não é de sua responsabilidade informar o que representam tais parâmetros em termos de recursos de rede (p.ex., quantidade de banda).

2.4 Considerações Finais

Neste capítulo, apresenta-se as principais abordagens que tratam de QoS, bem como os principais parâmetros de QoS relacionados à camada de aplicação e à camada de rede. Discute-se também uma revisão conceitual sobre as duas principais arquiteturas de provisão de QoS em redes IP. A Arquitetura DiffServ foi examinada em detalhes, permitindo desta forma, apresentar as suas vantagens e deficiências.

No próximo capítulo, componentes adicionais a Arquitetura DiffServ são apresentados, esses componentes em conjunto com os elementos da própria Arquitetura DiffServ tratam de alguns dos aspectos não contemplados na arquitetura e nem nos trabalhos anteriormente citados.

Capítulo 3

Descrição da Arquitetura Proposta

Neste capítulo apresenta-se um modelo cujos elementos são capazes de tratar alguns aspectos não tratados pela Arquitetura DiffServ, entre eles: **i.** o mapeamento dos parâmetros de QoS da camada de aplicação para os de rede; **ii.** negociação de uma banda de transmissão, ou seja, a aplicação é informada se pode ou não, ser atendida segundo a disponibilidade de alguma classe DS; **iii.** adaptação do fluxo na origem, segundo a sugestão de uma banda de transmissão disponível ou quando nenhuma banda de transmissão existente não atender as necessidades da aplicação; **iv.** configuração do domínio DiffServ para o enquadramento de um fluxo depois da negociação de uma banda de transmissão.

3.1 Aspectos Gerais da Arquitetura Proposta

O trabalho de Rossano [29] é o modelo para a replicação das funcionalidade e do desenvolvimento dos componentes: IQoS (*Interceptor QoS*), BB (*Bandwidth Broker*) e o TCD (*Traffic Control Daemon*) descritos na Fig. 3.1. Este trabalho adiciona ao trabalho do Rossano os componentes *Monitor-Adapter* e o Ambiente de Gerenciamento. O *Monitor-Adapter* monitora os fluxos de uma aplicação. Se uma aplicação não honrar o acordo, SLA contratada, o *Monitor-Adapter* sinaliza o BB para que ele tome uma ação segundo as regras de policiamento. A importância do *Monitor-Adapter* consiste em detectar o seguinte problema: se os fluxos não forem administrados e monitorados, aplicações que não respeitarem o contrato e submeterem a classe uma vazão maior que a contratada, comprometerá o serviço já oferecido a outro fluxo da mesma prioridade e classe. O Ambiente de Gerenciamento é adicionado para incluir as seguintes capacidades ao projeto: **i.** interface gráfica para que o administrador do domínio comunique com o BB e utilize todas as suas funcionalidades; **ii.** ambiente de fácil operação; **iii.** todas as informações relevantes a respeito das configurações das classes DS possa ser incluídas, visualizadas, alteradas ou removidas. Os diagramas utilizados para a modelagem dos componentes estão de acordo com a especificação UML (*Unified Modeling Language*) [31, 32].

Componente BB - *Bandwidth Broker*

Na Arquitetura IntServ, um novo fluxo é iniciado somente quando os recursos são previamente reservados. A reserva é feita através do protocolo RSVP. A função do RSVP é de

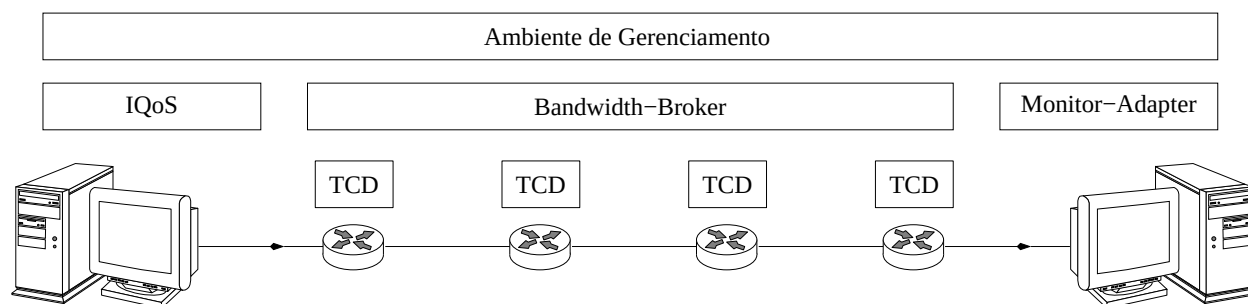


Fig. 3.1: Elementos da Arquitetura Proposta.

sinalizar todos os roteadores a reservarem, se possível, os recursos para um fluxo em particular. A Arquitetura DiffServ não especifica nenhum elemento para o controle de admissão, dando a abertura da inclusão deste componente.

Componente IQoS - *Interceptor QoS*

O IQoS é responsável pela tradução dos parâmetros de QoS da camada de aplicação para os parâmetros de rede. Após a tradução ele faz uma requisição de uma banda de transmissão para o BB. Com a utilização do IQoS, a aplicação não precisa se preocupar com todos os aspectos de requisição de banda entre eles, como é realizada a tradução e como requisitar um banda.

Componente *Monitor-Adapter*

Depois da criação de um contrato que especifica os parâmetros de QoS, o *Monitor-Adapter* verifica constantemente se a aplicação está ou não, respeitando o contrato. Quando uma violação é detectada, uma readaptação da rede ou do serviço deve ser realizada. Para o usuário, a readaptação da rede é melhor do que a alteração dos parâmetros de QoS escolhido por ele.

O *Monitor-Adapter* monitora os parâmetros da camada de rede relacionados com o serviço, com base nesses valores e nas políticas previamente acordadas, solicita a reconfiguração do domínio DiffServ. Se a reconfiguração não for possível, duas alternativas podem ser realizadas: **i.** o usuário será notificado e poderá fazer uma nova negociação; **ii.** os parâmetros dos serviços serão alterados para se adaptar a situação atual da rede.

Componente TCD - *Traffic Control Daemon*

A configuração do tratamento de um fluxo na rede é realizada através de ferramentas para manipular o comportamento do controle de tráfego em um domínio DiffServ. O processo manual é lento, em relação ao dinamismo da rede e está sujeito a erros. Para automatizar as configurações do controle de tráfego, o TCD está presente em todos os roteadores que formam o domínio DiffServ. Ele é a interface entre o controle de tráfego, que é específico do sistema operacional, e o componente BB. As ações executadas pelo TCD ocorrem segundo uma sinalização realizada pelo BB. O BB tem consciência do sucesso ou fracasso de uma ação

executada pelo TCD com o intuito de permitir que as estruturas de dados mantidas pelo BB fiquem consistentes com as ações executadas pelo TCD.

3.2 Especificação da Arquitetura Proposta

Nesta seção é apresentada uma descrição das funcionalidades dos componentes BB, IQoS, TCD e o Ambiente de Gerenciamento e suas interações. A caracterização deles consiste na descrição das interfaces IDL (*Interface Description Language*) dos objetos que os referenciam. A Fig. 3.1 apresenta estes componentes e suas localizações na Arquitetura DiffServ.

3.2.1 Componente BB

O componente BB concentra toda as informações da configuração do domínio DiffServ, ele é o ponto de acesso entre o Ambiente de Gerenciamento e os TCDs. Dentre as funcionalidades do BB destacam-se: **i.** controle de admissão; **ii.** negociação de uma banda de transmissão com o IQoS; **iii.** configuração das classes DS e configuração do enquadramento dos fluxos no domínio DiffServ. Para evitar inconsistências, o BB deve ser único em cada domínio DiffServ, para que o BB possa realizar estas tarefas cada TCD no domínio DiffServ deve se registrar no BB, fornecendo as características de suas interfaces de rede e sua localização dentro do domínio.

As operações suportadas pelo componente BB são (Fig. 3.2):

- `callBackAll()` - conecta com todos os TCDs registrados, a função deste método é verificar se os TCDs estão respondendo aos comandos do BB;
- `callcreateClassTcd()`: requisita o TCD para criar uma classe no controle de tráfego;
- `callcreateFilterTcd()`: requisita o TCD para criar um filtro no controlador de tráfego, com a finalidade de enquadrar um fluxo em uma classe DS;
- `createClass()`: permite o Ambiente de Gerenciamento requisitar a criação de classes no controlador de tráfego associado a alguma classe DS;
- `createClassesDefaultDSMARK()`: cria todas as classes do `tc` nos roteadores de borda, capaz de marcar o campo `DSFIELD` com um dos valores dos especificados pela Arquitetura DiffServ. Este método automaticamente cria estas classes do `tc` somente nos TCDs de borda. Esta operação ajuda no processo de configuração do domínio.
- `createFilter()`: requisita a criação de filtros nos TCDs que para permitir o enquadramento dos fluxos de uma determinada aplicação a um serviço DiffServ;
- `freeUseofClass()`: permite a aplicação informar o seu desejo de não mais necessitar de uma banda de transmissão;
- `getAccount()`: retorna a situação do uso de cada classe DS;
- `getAllClasses()`: devolve todas as classes criadas em um TCD específico;

classBB
+ arrayaccount: classArrayAccount + vectorTCD: Vector
+ callBackAll() + callcreateClassTcd() + classcreateFilterTcd() +.createClass(): classClass + createClassesDefaultDSMARK(String): int + createFilter(String, FilterDS): int + createFilterDSMARK(int, RequestOfBandWidth): int + freeUseofClass(int, int): int + getAccount(): Account[] + getAllClasses(String): classDS[] + getAllFilters(String): filterDS[] + getCosofClassid(String, String): int + getDevofTCD(String): String + getPosTcdsCore(): int + getPosTcdsEdge(): int + getPosvectorTCD(String): int + getQtdClasses(String): int + getQtdClasses(String): int + getQtdFilters(String): int + getVectorTCD(): Vector + getVectorTCDString(): String[][][] + nextUnderBand(int): int + propagateConfTcd(String): int + removeClass(String, ClassDS): int + removeFilter(String, FilterDS): int + removeFilterDSMARK(int, RequestOfBandWidth): int + requestClass(RequestOfBandWidth): int + setRegister(String, String, String): int + setUNRegister(String): int

Fig. 3.2: Propriedades e Métodos do BB.

- **getAllFilters()**: permite a visualização de todas as aplicações enquadradas em uma determinada classe DS;
- **getCoSofClassid()**: retorna o valor da classe DS que está associada a uma classe do controlador de tráfego em um nó DiffServ;
- **getDevofTCD()**: retorna a interface de rede a qual o TCD está manipulando;
- **getPosTCDsCore()**: retorna todos os TCDs de núcleo;
- **getPosTCDsEdge()**: retorna todos os TCDs de borda;
- **getPosvectorTCD()**: permite obter informações de um TCD em sua base de informações;
- **getQtdClasses()**: retorna a quantidade de classes DS criadas no controlador de tráfego;
- **getQtdFilters()**: retorna a quantidade de filtros criados no controlador de tráfego;

- **getVectorTCD()**: faz referência a base de informações dos registros de todas as ações do gerenciamento do domínio DiffServ;
- **getVectorTCDString()**: retorna todos os TCDs registrados no domínio DiffServ;
- **nextUnderBand()**: procura uma banda onde o valor seja mais próximo do requisitado;
- **propagateConfTcd()**: o domínio DiffServ é formado por um conjunto de roteadores que compartilham as mesmas configurações de classes, assim o domínio poderá fornecer um único comportamento a um tráfego fim a fim. A configuração de uma classe através do Ambiente de Gerenciamento é facilitado por este ambiente gráfico construído. Mesmo assim, o administrador do domínio poderá configurar os roteadores diferentemente, ferindo a especificação e gerando algum problema na prestação do serviço. Este método tem uma função primordial de propagar uma configuração de um TCD específico para os demais TCDs de núcleo. A propagação é um processo de clonagem das configurações atuais de um TCD, após esta operação todos os TCDs de núcleo terão as mesmas classes e filtros;
- **removeClass()**: remove uma classe DS e por consequência requisita os TCDs para configurar o domínio DS devido a esta remoção;
- **removeFilter()**: remove o enquadramento de um fluxo associado a uma classe DS;
- **requestClass()**: procura alguma classe que tenha recursos para atender a requisição de uma banda de transmissão. Para realizar esta tarefa, este método procura classe por classe. Se alguma classe for encontrada é necessário pesquisar sua utilização, afim de descobrir se ela esta sendo usada e quanto dela está sendo usada;
- **setRegister()**: para que o BB possa saber quais os TCDs presentes no domínio DiffServ é necessário que os TCDs invoquem este método quando eles são executados, a execução dos TCDs é um processo manual;
- **setUnRegister()**: toda vez que um TCD é finalizado, ele chama este método para que o BB tenha a lista dos TCDs presentes no domínio DiffServ atualizadas.

A interface BB possui a seguinte descrição IDL:

```
interface BB {
    long setRegister(in string TCDID, in string TYPETCD, in string device);
    long setUNRegister(in string IDTCD);
    void callBackAll();
    string_tridimensional getVectorTCDString();
    string getDevofTCD(in string TCDID);
    long getQtdClasses(in string TCDID);
    long getQtdFilters(in string TCDID);
    ClassesDS getAllClasses(in string TCDID);
    FiltersDS getAllFilters(in string TCDID);
    long createClass(in string TCDID, in ClassDS classds);
}
```

```

    long removeClass(in string TCDID, in ClassDS classds);
    long createFilter(in string TCDID, in FilterDS filter);
    long removeFilter(in string TCDID, in FilterDS filter);
    long requestClass(in RequestOfBandWidth requestbw);
    long createClassesDefaultDSMARK(in string TCDID);
    long removeFilterDSMARK(in long flowid, in RequestOfBandWidth rqstbw);
    Accounts getAccount();
    long freeUseofClass(in long classe, in long free);
    long getCosofClassid(in string tcdid, in string classid);
    long nextUnderBand(in long rqstbw);
    long propagateConfTcd(in string tcdid);
};

```

O Diagrama de classes dos elementos que compõem o BB são apresentados na Fig. 3.3:

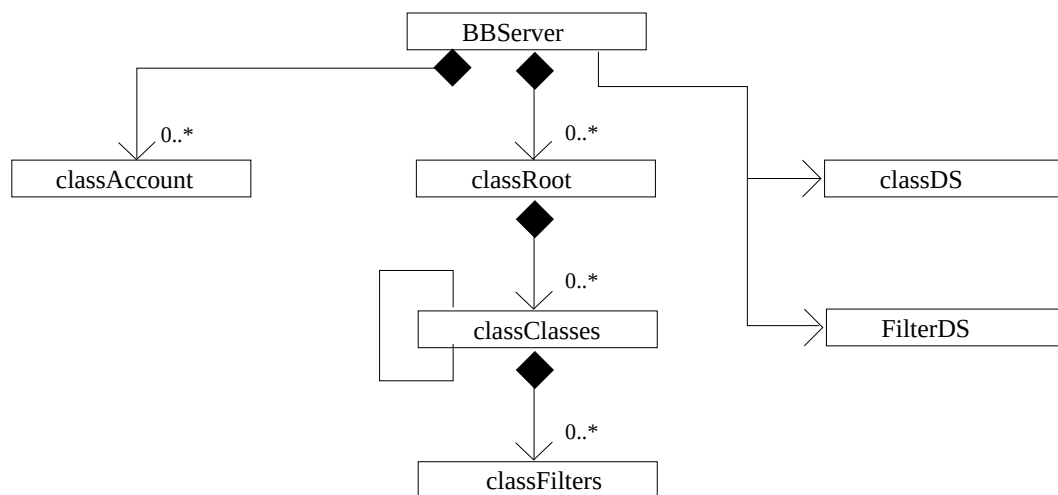


Fig. 3.3: Elementos que Compõem o Componente BB.

A classRoot

Para cada TCD registrado, o BB mantém uma classe **classRoot** para modelar as informações associadas ao TCD. A classe **classRoot** (Fig. 3.4) é caracterizada pelos métodos descritos em sequência:

- **addClasse()**: atribui uma classe do tipo **classClasse** a propriedade classe. A **classClasses** modela uma classe de alguma disciplina de fila associada a uma classe DS;
- **findClassid()**: toda classe possui um identificador nomeado por **classid**. Este método procura por toda a estrutura hierarquica uma classe com **classid** igual ao parâmetro **classid**;
- **getClasse()**: retorna um objeto do tipo **classClasses**;
- **getQtdClasses()**: retorna a quantidade de objetos do tipo **classClasses**;

classRoot
+ classe : classClasses + namedevice : String - tcdid : String - type : String
+ classRoot() + addClasse(classClasses): boolean + findClassid(String): boolean + getClasse(): classClasses + getNameDevice(): String + getQtdClasses(): int + getQtdFilters(): int + getTcdid(): int + getType(): String + getVectorwithallClasses(): Vector + getVectorwithallFilters(): Vector + hasBandWidth(int): classAccountClass + hasClasse(): boolean + isClassEmpty(): boolean + isCore(): boolean + isEdge(): boolean + removeClass(String): int + removeFilter(FilterDS): int + searchFather(String): classClasses + setNameDevice(String): void + setTcdid(String): void + setType(String): void

Fig. 3.4: Diagrama da Classe classRoot.

- **getQtdFilters()**: retorna a quantidade de objetos do tipo **classFilter** que armazenam as informações dos enquadramentos criados no controlador de tráfego;
- **getTcdid**: retorna o identificador do TCD;
- **getType()**: retorna o tipo do TCD, ou seja, se ele é de borda ou de núcleo;
- **getDev()**: retorna o dispositivo que o TCD está manipulando;
- **getVectorwithallClasses()**: retorna todos os objetos do tipo **classClasses** criados;
- **getVectorwithallFilters()**: retorna todos os objetos do tipo **classFilter** incluídos;
- **hasBandWith()**: verifica se existe uma classe DS com uma banda maior ou igual a requisitada;
- **hasClasse()**: verifica se um objeto do tipo **classClasses** foi criado;
- **isCore()**: retorna verdadeiro se o TCD é de núcleo no domínio;
- **isEdge()**: retorna verdadeiro se o TCD é de borda no domínio;
- **removeClass()**: remove uma classe DS;
- **removeFilter()**: remove um filtro no controlador de tráfego usado para enquadrar uma aplicação a uma classe DS;

- `searchFather()`: retorna o objeto do tipo `classClasses` hierarquicamente superior;
- `setNameDevice()`: configura o nome da interface de rede que o TCD manipula;
- `setTcdid()`: configura o identificador do TCD;
- `setType()`: configura o tipo do TCD, ou seja, se ele é de borda ou de núcleo.

A `classClasses`

Antes de descrever as funcionalidades desta classe, é importante mostrar os três tipos diferentes de classes que este trabalho referencia. O primeiro tipo é a classe que está associada a uma disciplina de fila disponível no controlador de tráfego, este tipo de classe serve para acomodar um fluxo, e aplicar a este fluxo uma ação que pode ser: atrasar os pacotes pertencente ao fluxo, descartar pacotes do fluxo e realizar a marcação do campo DSFIELD de um pacote IP, sempre que neste texto a classe citada for deste tipo, o termo classe do controlador de tráfego ou classe do `tc` será usado. O `tc` é uma ferramenta para manipular o controle de tráfego no Linux. O segundo tipo, está relacionado com as classes da Arquitetura DiffServ, quando neste texto a classe citada for deste tipo, o termo classe DS será usado. O terceiro tipo refere-se a classe usada para modelar um objeto computacional criado em uma linguagem de programação, neste caso o termo utilizado será simplesmente classe.

A `classClasses` (Fig. 3.5), assim como a `classRoot`, armazenam as informações relacionadas com os TCDs espalhados no domínio DS. Esta classe possui uma coleção de classes do mesmo tipo, uma vez que elas são criadas hierarquicamente. O sistema não limita criação de classes associadas ao controlador de tráfego. As classes filhas são incluídas em um vetor disponível na própria classe. O mesmo acontece com os filtros, já que ela também tem a propriedade `filters` que é uma coleção de objetos do tipo `classFilters`. A classe `classFilter` é explicada na Seção 3.2.1.

A classe `classClasses` é caracterizada pelos métodos descritos em sequência:

- `addClasses()`: adiciona um objeto do tipo `classClasse` em sua coleção de classes;
- `addFilter()`: adiciona um objeto do tipo `classFilter` na sua coleção de filtros;
- `autoaddClasses()`: este método tem a função de adicionar um objeto do tipo `classClasses` criada por ela mesma em sua coleção de objetos;
- `autoaddFilters()`: adiciona todos os objetos do tipo `classFilter` em sua coleção de filtros;
- `getClasses()`: retorna a referência do objeto do tipo `classClasses`;
- `getClassid()`: retorna o identificador da classe criada no controlador de tráfego;
- `getCos()`: retorna a classe DS associada a esta classe;
- `getFather()`: retorna a classe hierarquicamente superior;
- `getFilters()`: retorna os objetos do tipo `classFilter` utilizados para enquadrar os fluxos;
- `getOptions()`: devolve os parâmetros da classe do controlador de tráfego associado a esta classe;
- `getSizeVector()`: retorna a quantidade de classes incluídas em sua coleção;
- `hasBandWidth()`: retorna uma classe que tenha uma banda maior ou igual ao valor requisitado;

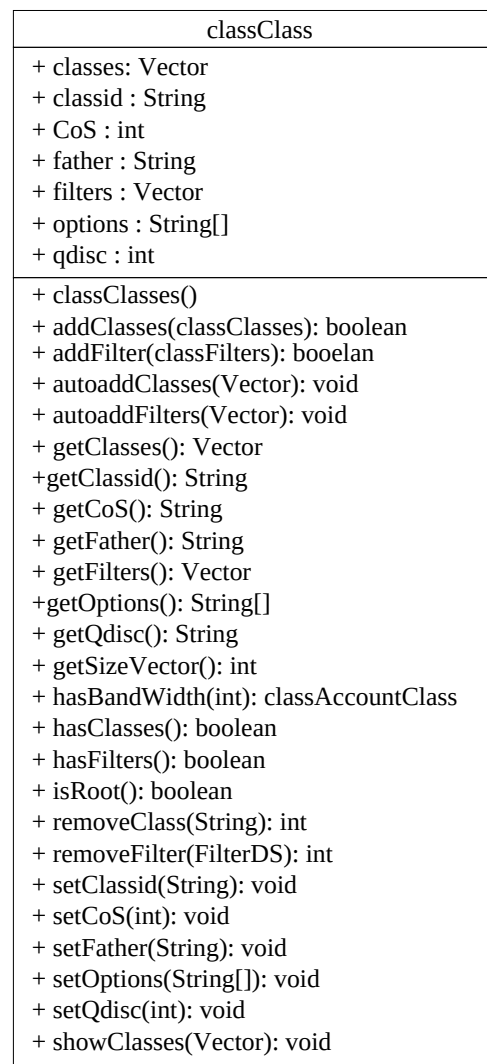


Fig. 3.5: Diagrama da Classe classClass.

- **hasClasses()**: retorna verdadeiro se existir algum objeto na coleção de classes;
- **hasFilters()**: retorna verdadeiro se existir algum objeto na coleção de filtros;
- **removeClasses()**: remove um objeto do tipo **classClasse** de sua coleção;
- **removeFilter()**: remove um objeto do tipo **classFilter** de sua coleção;
- **setClassid()**: configura o identificador da classe;
- **setCos()**: configura qual classe DS será associada a esta classe;
- **setFather()**: configura qual é a classe pai;
- **setOptions()**: configura os parâmetros para criar uma classe no controlador de tráfego;

- **setQdisc()**: configura o tipo da disciplina de fila associada a classe do controlador de tráfego;
- **showClasses()**: retorna um vetor de todos os objetos do tipo **classClasses** incluídos.

A classFilter

A **classFilter** (Fig. 3.6) tem a função de modelar dois tipos de filtros, o U32 e o TCINDEX. O U32 utiliza os campos do cabeçalho IP, UDP e TCP para destinar um pacote a uma classe no controlador de tráfego. O TCINDEX utiliza o campo DSFIELD para descobrir a classe que o pacote será destinado.

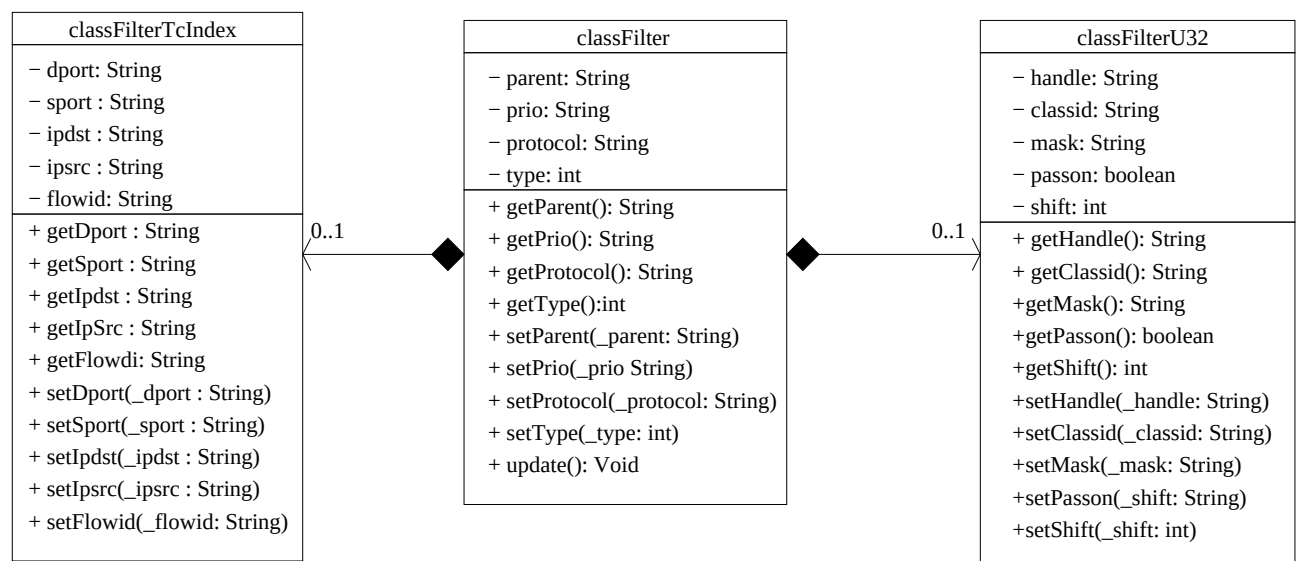


Fig. 3.6: Diagrama da Classe classFilter.

3.2.2 Componente IQoS

O IQoS (Fig. 3.7) retira da aplicação a responsabilidade de traduzir os parâmetros de QoS da aplicação para os de rede. Adicionalmente, ele tem a função de requisitar uma banda de transmissão com o BB. A resposta do BB a uma requisição pode ser de dois tipos, a primeira, é uma confirmação de que o domínio DS possui recursos, e a segunda, é que não há recursos suficientes. A resposta do segundo tipo, traz um valor que corresponde a uma banda de transmissão menor do que a requerida, caso exista alguma. A aplicação pode diminuir a suas exigências e fazer uma nova negociação. Caso ela não faça uma negociação ou uma renegociação, os fluxos desta aplicação receberão o tratamento de melhor esforço pela rede. O fato da rede não suportar as exigências de QoS, não significa que os fluxos não possam ser criados, e sim, que a aplicação não terá qualquer garantia da rede.

Quando uma aplicação não necessita mais da reserva, previamente realizada, o IQoS deverá sinalizar o BB para desfazer a reserva. Este processo consiste em remover o enquadramento dos fluxos no domínio DiffServ e atualizar a tabela de contabilidade de uso das classes DS. O componente IQoS é caracterizado pelos métodos descritos em sequência:

- **availableBandWidth()**: procura por uma banda de transmissão para acomodar o fluxo;

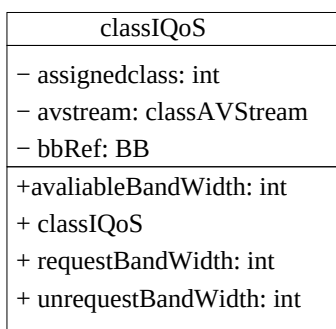


Fig. 3.7: Propriedades e Métodos do IQoS.

- **requestBandWidth()**: requisita do BB uma classe DS para acomodar suas exigências quanto à QoS. O BB faz a procura analisando a banda de transmissão associada a classe DS e na tabela de contabilidade do uso das classes DS;
- **unrequestBandWidth()**: informa ao BB para remover o enquadramento dos fluxos e atualizar a contabilidade do uso de uma classe DS.

O diagrama de classes da Fig 3.8 apresenta algumas classes para modelar objetos com a função de armazenar as características de mídias, de áudio e vídeo, acrescentadas ao IQoS.

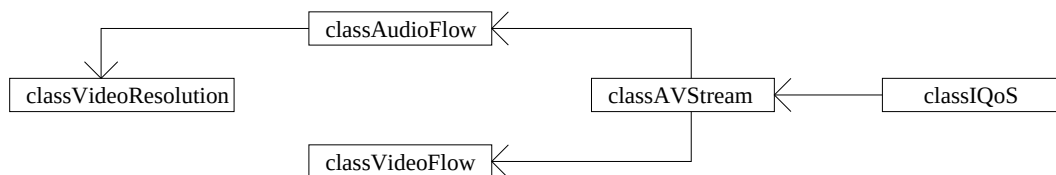


Fig. 3.8: Elementos que Compõem o Componente IQoS.

A classe classAVStream

A classe **classAVStream** (Fig. 3.9) modela os parâmetros de QoS da camada de aplicação relacionados com os fluxos de áudio e de vídeo. O objetivo desta classe é armazenar valores que possibilitem descobrir qual a quantidade de banda necessária para atender a aplicação. Devido à dificuldade computacional de agregar as funcionalidades deste modelo a uma aplicação legada, esta classe permite criar objetos que simulem o comportamento de uma aplicação multimídia. Todos os parâmetros associados ao áudio e ao vídeo são utilizados pela ferramenta RUDE na geração de tráfego na rede.

A classAudioFlow

A função da classe **classAudioFlow** (Fig. 3.10) é de modelar os fluxos de áudio, suas propriedades correspondem as características do áudio, a seguir elas são listadas:

- **samplesize**: valor da taxa de amostragem do som. Ex.: 44.000Hz;
- **samplerate**: quantidade de *bits* para armazenar os valores capturados;

classAVStream
- arrayAudioFlow: Vector - arrayVideoFlow: Vector - ipsrc: String - ipdst: String - sport: int - dport: int - duration: int
+ addAudioFlow(audioflow: classAudioFlow): void + addVideoFlow(videoflow: classVideoFlow): void + getArrayAudio(): Vector + getArrayVideo(): Vector + getIpsrc(): String + getIpdst(): String + getDport(): int + getSport(): int + getDuration(): int + setIpsrc(ipsrc: String): void + setIpdst(ipdst: String): void + setSport(sport: int): void + setDport(dport: int): void + setDuration(duration: int): void + getSunLenghBandWidth(): int

Fig. 3.9: Diagrama da Classe classAVStream.

- `nunchannel`: quantidade de canais de som;
- `quantization`: tipo de codificação do áudio.

A classVideoFlow

A classe `classVideoFlow` (Fig. 3.11) é usada para modelar os fluxos de vídeo, as propriedades desta classe correspondem as características de um vídeo. As propriedades são:

- `colordepth`: quantidade de *bits* para armazenar uma cor;
- `colormode`: o tipo do mapa de cores;
- `samplerate`: quantidade de *frames* por segundo;
- `resolution`: armazena os valores da dimensão do vídeo.

A classVideoResolution

A classe `classVideoResolution` (Fig. 3.12) é usada pela `classVideoFlow` para modelar as dimensões de um vídeo. As suas propriedades são:

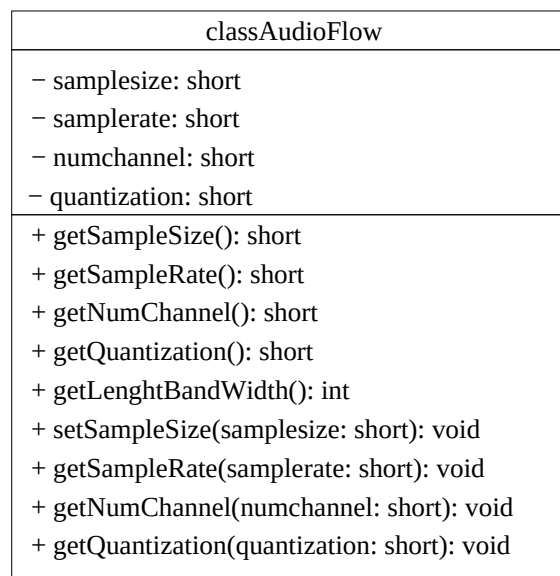


Fig. 3.10: Diagrama da Classe classAudioFlow.

- **width**: a largura da imagem;
- **height**: a altura da imagem.

Atividades da Classe IQoSTest

Para que as aplicações possam usufruir das funcionalidades incorporadas por este modelo é necessário que elas possam interagir com o IQoS. A finalidade de um objeto do tipo **IQoSTest** é simular uma aplicação produtora de áudio e vídeo e que tenha acesso ao IQoS. A requisição de alguma banda é uma atividade que geralmente é iniciada pela aplicação cliente. Neste exemplo, quem requisita é a aplicação que está produzindo os fluxos. O IQoS pode ser utilizado tanto por uma aplicação cliente para garantir uma banda de recepção ou por uma aplicação servidora para garantir uma banda de transmissão, os dois casos são possíveis, pois é passado pela aplicação para o IQoS os parâmetros de rede: endereço de origem, endereço de destino, porta UDP/TCP de origem e porta UDP/TCP de destino, para distinguir estes dois casos. é necessário informar devidamente esses parâmetros para que o enquadramento dos fluxos seja feito corretamente. Na Fig. 3.13, o diagrama de atividade apresenta o processo interno do objeto **IQoSTest**.

1. primeiro este objeto cria um objeto do tipo **classAudioFlow** e outro objeto do tipo **classVideoFlow**;
2. configura os parâmetros dos objetos criados;
3. cria um objeto do tipo **classIQoS**;
4. informa ao IQoS os parâmetros de QoS;
5. o IQoS requisita uma banda de transmissão;
6. se o BB possuir alguma classe DS que suporte a requisição, este objeto gera um tráfego na rede, senão a próxima banda disponível é informada pelo BB através do IQoS.

classVideoFlow
– colordepth: int – colormode: int – samplerate: int – resolution: classVideoResolution
+ getColorDepht(): int + getColorMode(): int + getSampleRate(): int + getVideo_Resolution(): classVideoResolution + setColorDepht(colordepht: int) + setColorMode(colormode: int) + setSampleRate(samplerate: int) + setResolution(videoresolution: classVideoResolution) + getLenghBandWidth(): int

Fig. 3.11: Diagrama da Classe classVideoFlow.

classVideoResolution
– height: short – widht: short
+ getHeight(): short + getWidth(): short + setHeight(height: short): void + setWidth(widht: short): void

Fig. 3.12: Propriedades e Métodos da Classe classVideoResolution.

3.2.3 Componente TCD

A função do TCD (Fig. 3.14) é tornar dinâmico o processo de configuração manual de todos os roteadores do domínio DiffServ. Os TCDs são modelados pela classe `classTCD` e suas operações são realizadas somente após uma requisição do BB. Os métodos que o caracterizam são:

- `getTcid()`: retorna a identificação do TCD;
- `getType()`: retorna se o TCD é de borda ou de núcleo;
- `setTcid()`: configura a identificação do TCD;
- `setType()`: atribui um novo valor a propriedade `type`;
- `createClass()`: cria uma classe no controlador de tráfego. A criação é de fato feita através de uma chamada de sistema do sistema operacional;
- `createFilter()`: cria um filtro associado a uma classe no controlador de tráfego;
- `removeClass()`: remove uma classe cujos valores são iguais aos valores passado por parâmetro;

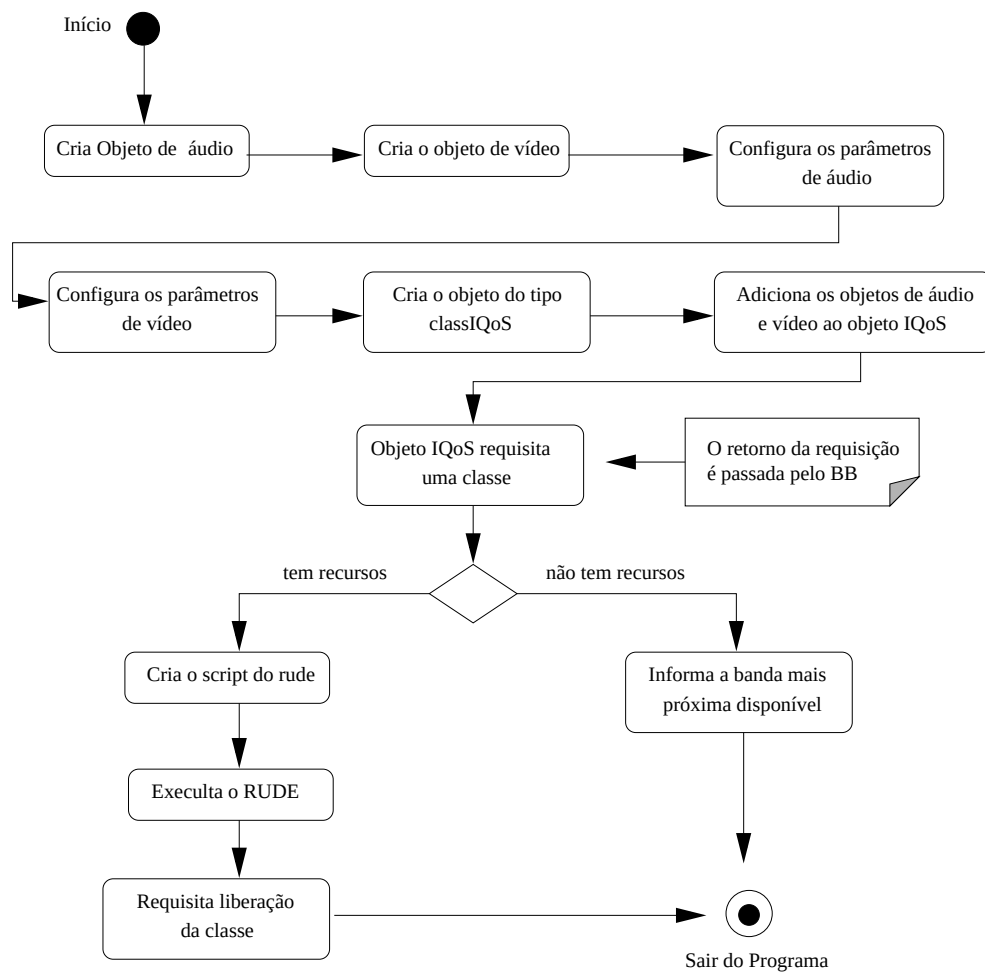


Fig. 3.13: Diagrama de Atividades do Objeto IQoSTest.

- **removeFilter()**: remove um filtro cujos valores estão iguais aos valores passado por parâmetro;
- **unRegister()**: este método é chamado pela interceptação do sinal de finalização do programa, sua função é de requisitar que o BB o desregistre;
- **getContactBB()**: permite testar a comunicação com o BB.

Os métodos responsáveis pela criação e remoção das classes e dos filtros, utilizam os valores passados por parâmetros e constrói uma expressão, seguindo as regras sintáticas do comando `tc` [33]. Os erros da execução do `tc` são obtidos pelo valor de retorno da chamada de sistema ao método `runCommand()` da classe `classUtils`, e este por sua vez, retorna o valor do erro para o BB. O BB com o número do erro toma alguma ação com o objetivo de manter consistente os TCDs com a sua base de informação.

Na Fig. 3.15 as classes `ClassDS` e `FilterDS` são utilizadas para a troca de dados entre os componentes Ambiente de Gerenciamento, BB e o TCD. A classe `classUtils` é detalhada na seção 3.2.3.

A `classTCD` não mantém nenhuma estrutura de dados a respeito das classes e filtros associados ao controlador de tráfego por ela criada, reforçando a sua conformidade com o modelo proposto.

A Interface TCD possui a seguinte descrição IDL:

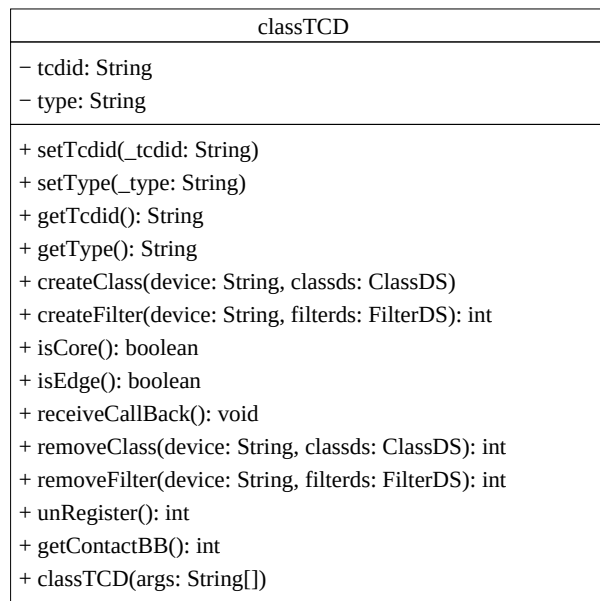


Fig. 3.14: Propriedades e Métodos da Classe classTCD.

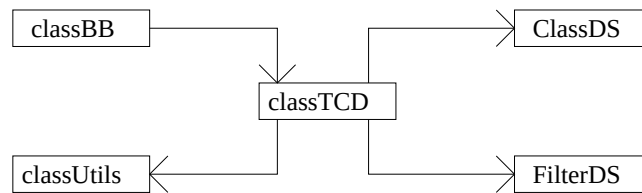


Fig. 3.15: Elementos que Compõem o Componente TCD.

```

interface TCD {
    boolean isCore();
    boolean isEdge();
    long createClasses(in string Device, in ClassDS classds);
    long removeClasses(in string Device, in ClassDS classds);
    long createFilters(in string Device, in FilterDS filterds);
    long removeFilters(in string Device, in FilterDS filterds);
    void receiveCallBackBB();
};
  
```

A classUtils

A classe `classUtils` (Fig. 3.16) modela as funcionalidades que são utilizadas por vários objetos que representam os componentes do modelo. Para evitar a duplicação de código e ganhando assim em consistência e reutilização dos mesmos esta classe foi criada. As propriedades desta classe, são constantes que representam as várias classes DS padronizadas pela especificação da Arquitetura DiffServ e constantes que identificam os tipos de filtros e disciplinas de filas do controlador de tráfego.

As operações suportadas por esta classe se encarregam das seguintes funcionalidades:

classUtils
– vectorDSCP: String[] + AF11: final int + AF12: final int + AF13: final int + AF21: final int + AF22: final int + AF23: final int + AF31: final int + AF32: final int + AF33: final int + AF41: final int + AF42: final int + AF43: final int
+ FILTER_TCINDEX_VALUE: final int + FILTER_U32_VALUE: final int + QDISC_DSMARK_VALUE: final int + QDISC_GRED_VALUE: final int + QDISC_HTB_VALUE: final int + getFlowid(dscp :int): String + getPOs(DSCP: String): int + getValueIntDSCP(DSCP: String): int + getValueStringDSCP(DSCP: String): String + getValueStringDSCP(pos: int): String + getbps(nr: int, unit: String): int + initialValuesOfFilter(filterds: FilterDS): void + runCommand(COMMAND: String):int

Fig. 3.16: Propriedades e Métodos da Classe classUtils.

- **getFlowid()**: retorna a identificação de um fluxo associado a um classe DS;
- **getValueIntDSCP()**: retorna a classe DS de um DSCP;
- **getValueStringDSCP()**: retorna a classe DS, no formato de uma *string*, de um DSCP;
- **getbps()**: retorna um número em bps;
- **initialValuesOfFilter()**: inicia os valores de um objeto da classe **FilterDS**, utilizada para troca de informações entre os componentes;
- **runCommand()**: faz uma chamada de sistema para que o sistema operacional execute um comando passado como parâmetro.

3.2.4 Componente Ambiente de Gerenciamento

O Ambiente de Gerenciamento (Fig. 3.17) é uma ferramenta gráfica disponível em uma interface Web, que permite ao administrador, configurar o domínio DiffServ em qualquer máquina que possua um

navegador para a Internet (*Browser*). Este componente representa a interface com o BB e possui uma lista das funções que podem ser executadas pelo BB. Dentre estas funcionalidade este componente se preocupa com a apresentação dos dados de uma forma legível. Outro aspecto é a agilização na configuração do domínio DS.

FrmAmbGer
<pre> + exitForm(): void + jbtActionConnectionBB(): void + jbtActionCreateClasses(): void + jbtActionCreateFilters(): void + jbtActionDisconnectBB(): void + jbtActionExit(): void + jbtActionRemoveClasses(): void + jbtActionRemoveFilters(): void + jbtActionShowClasses(): void + jbtActionShowFilters(): void + jbtActionShowTCDs(): void + jbtActionChangeUse(): void + jbtActionCreateClassesDefaultsTCDEdge(): void + jbtActionRefreshAccountUse(): void + jbtActionPropagatedConfTCDNucleo(): void </pre>

Fig. 3.17: Propriedades e Métodos da Classe FrmAmbGer.

O Ambiente de Gerenciamento é moldado pela classe **FrmAmbGer** que é uma generalização da classe **JApplet** do pacote `javax.swing` da linguagem Java. As operações suportadas por esta classe se encarregam das seguintes funcionalidades:

- **connectionBB()**: conecta com o BB;
- **createClasses()**: cria uma classe segundo as informações do objeto do tipo **FrmClasses**;
- **createFilters()**: requisita a criação de um filtro com base nas informações providas pelo objeto do tipo **FrmFilters**;
- **disconnectBB()**: desconecta do BB;
- **removeClasses()**: requisita a remoção de uma classe;
- **removeFilters()**: requisita a remoção de um filtro;
- **showClasses()**: atualiza a lista de classes criada;
- **showFiltes()**: atualiza a tabela de filtros;
- **showTCDs()**: atualiza a lista de TCDs inicializado no domínio;
- **changeUse()**: altera o uso de banda de transmissão associada a uma determinada classe DS;

- **createClassesDefaultTCDEdge()**: nos roteadores de borda a disciplina de fila utilizada é a DSMARK, uma vez que os valores do DSCP são padronizados, o processo de configuração dos classificadores da Arquitetura DiffServ é idêntico todas as vezes. Este método cria as classes associadas a disciplina de fila DSMARK responsáveis pela marcações dos pacotes;
- **refreshAccountUse()**: atualiza a tabela de uso das classes;
- **propagatedConfTCDNucleos()**: após a configuração de um TCD de núcleo, este método permite copiar o seu perfil para todos os outros TCDs de núcleo no domínio DiffServ.

3.3 Interações entre os Componentes da Arquitetura Proposta

Nesta seção, é apresentado o comportamento dinâmico dos componentes. As interações detalham os relacionamentos entre eles e a importância destes no projeto. As interações são apresentadas por meio de um conjunto de cenários que vão dos mais simples até os mais complexos e que descrevem as funcionalidades que os componentes propostos agregaram à Arquitetura DiffServ.

3.3.1 Listagem dos Componentes TCDs Disponíveis no Domínio DS

Este cenário, ilustra os primeiros passos que o administrador do domínio DiffServ faz quando os componentes são criados. A interface que o administrador utiliza para configurar o domínio DiffServ é o Ambiente de Gerenciamento são apresentadas no Capítulo 5. Na Fig. 3.18 são mostrados três objetos, o objeto associado ao componente Ambiente de Gerenciamento é do tipo **FrmAmbGer**, o objeto associado com o componente BB é do tipo **BBServer**, e o último, o objeto do tipo **TCDTest** associado ao componente TCD.

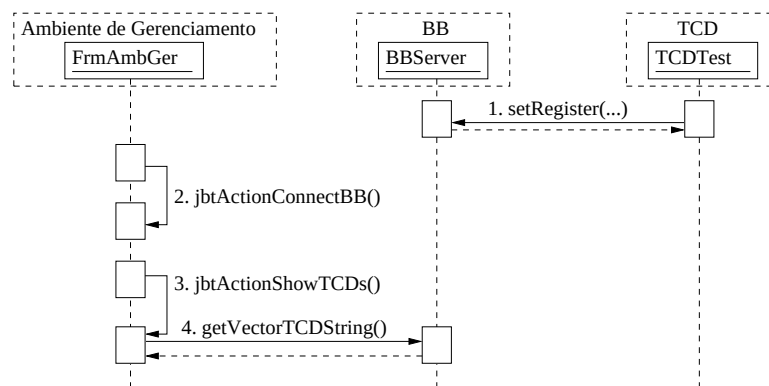


Fig. 3.18: Listagem dos TCDs Disponíveis no Domínio DS.

A listagem dos componentes TCDs é realizada toda vez que se quer saber quantos e quais são os TCDs no domínio DiffServ. A sequência para listar os TCDs é descrita no diagrama da Fig. 3.18.

1. o TCD quando criado em algum nó do domínio DiffServ registra-se no BB;
2. o Ambiente de Gerenciamento conecta com o BB;

3. o administrador requisita ao Ambiente de Gerenciamento para buscar todos os TCDs;
4. o ambiente de gerenciamento requisita os TCDs para o BB, em resposta a esta requisição, o BB retorna uma lista com todos os TCDs registrados.

3.3.2 Criação de uma Classe de Serviço DS

A Fig. 3.19 apresenta a criação de uma classe no controlador de tráfego pelo Ambiente de Gerenciamento, para esta tarefa é necessário os objetos: **FrmAmbGer**, **BBServer** e o **TCDTest**.

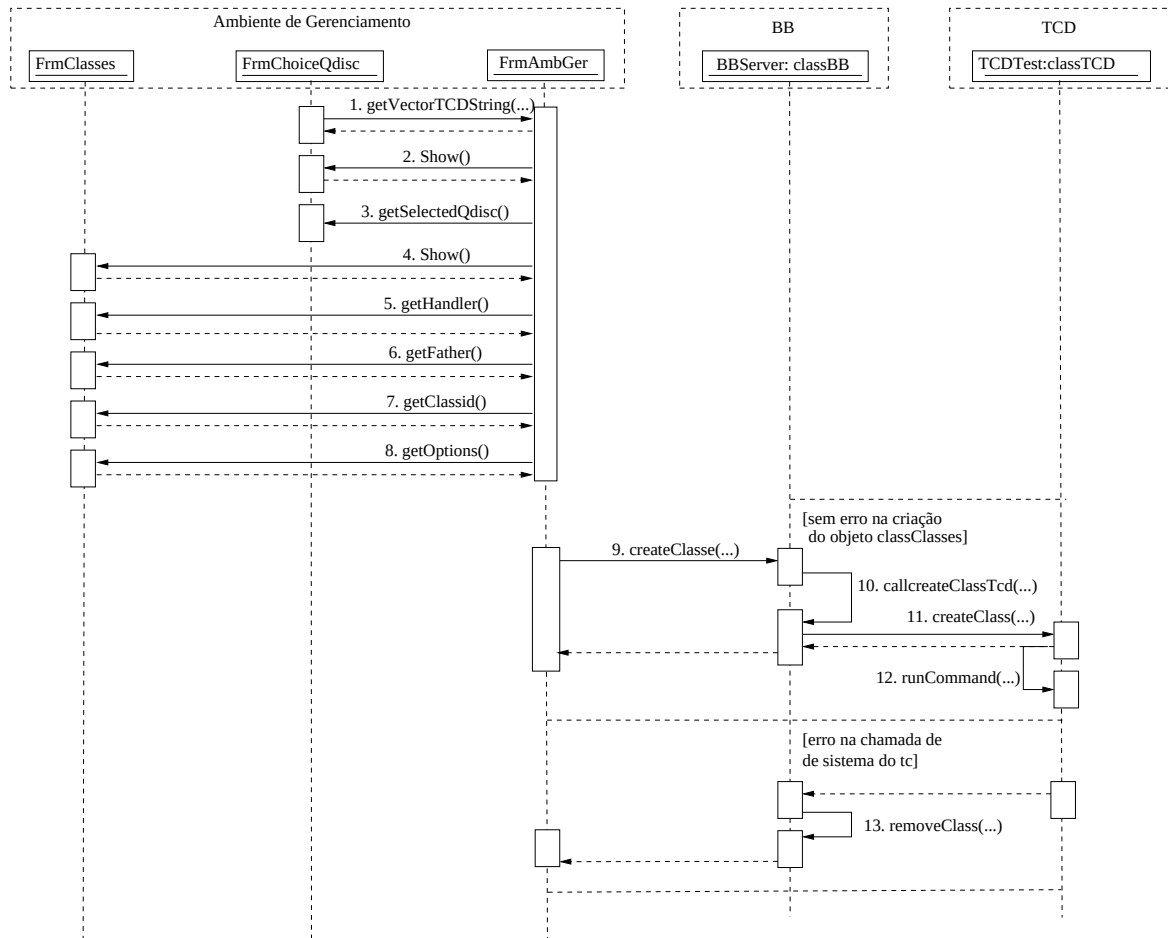


Fig. 3.19: Criação de uma Classe DiffServ.

1. obter a lista de todos os TCDs no domínio DiffServ;
2. mostra a tela para escolha do tipo de disciplina de fila associada a classe do controlador de tráfego, toda classe DS possui uma associação com uma classe do controlador de tráfego;
3. obter a disciplina de fila escolhida;
4. criar a tela para a configurar os parâmetros da classe;
5. desta sequência até a sequência 9, configura-se os parâmetros para criar a classe;

9. requisita ao BB para criar uma classe no TCD;
10. o BB armazena as informações da classe a ser criada, caso todos os parâmetros estejam corretos, o BB requisita ao TCD para criar uma classe no controlador de tráfego;
11. o BB requisita a criação da classe do controlador de tráfego para o TCD;
12. o TCD faz uma chamada de sistema para criar a classe;
13. caso algum erro na criação da classe pelo TCD ocorre, o BB remove as informações referente a classe e envia um valor de erro ao Ambiente de Gerenciamento.

3.3.3 Especificação e Requisição de uma Banda de Transmissão

A simulação de uma aplicação que requisita uma banda é feita através do objeto do tipo `IQoSTest`, este objeto simula um fluxo de áudio e/ou vídeo. A solicitação é realizada pelo componente IQoS que é especificado pela classe `classIQoS` e declarada pela própria classe `IQoSTest`. A sequência para especificar e requisitar uma banda de transmissão é apresentada na Fig. 3.20.

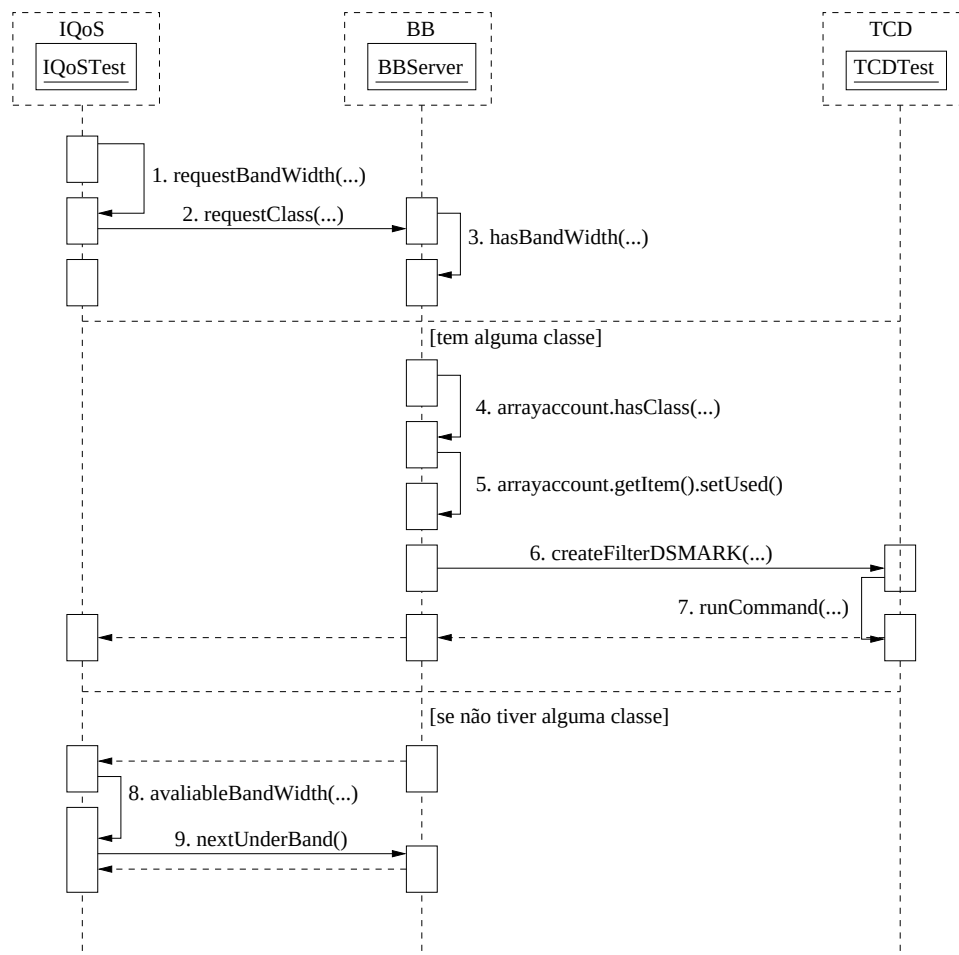


Fig. 3.20: Especificação e Requisição de uma Banda de Transmissão.

O conjunto de eventos que caracterizam a requisição de uma banda são descritos a seguir:

1. requisitar uma banda de transmissão para o IQoS;
2. o IQoS traduz os parâmetros de QoS da camada da aplicação para a camada de rede, e requisita uma banda;
3. verificar uma banda disponível, se tiver, a classe DS associada a essa banda é retornada;
4. verificar o uso da classe DS na tabela de contabilidades;
5. atualizar a tabela de contabilidade com mais esta nova requisição;
6. requisitar o TCD para enquadrar os fluxos da aplicação na classe DS disponível;
7. o TCD configura o domínio DiffServ;
8. retornar uma banda de transmissão mais próxima da requisitada;
9. a aplicação requisita uma outra banda de transmissão que esteja disponível.

3.3.4 Enquadramento de um Fluxo em uma Classe de Serviço DS

O enquadramento de um fluxo a uma classe DS é realizado através dos filtros no controlador de tráfego, para que isto aconteça é utilizado os seguintes objetos: `FrmAmbGer`, `BBServer` e o `TCDTest` (Fig. 3.21). A seqüência de eventos para realizar o enquadramento de um fluxo são a seguir enumeradas:

1. pedir para visualizar as classes DS criadas;
2. requisitar o BB para retornar todas as classes criadas em um TCD;
3. criar a tela para a configurar os parâmetros do filtro;
4. desta seqüência até seqüência 7, configura-se os parâmetros comuns entre os dois tipos de filtros o `U32` e o `TCINDEX`;
8. desta seqüência até a seqüência 12, configura-se os parâmetros associados com filtro `U32`;
13. desta seqüência até a seqüência 17, configura-se os parâmetros associados com o filtro `TCINDEX`;
18. o Ambiente de Gerenciamento requisita ao BB criar o filtro.
19. o BB armazena as informações do filtro em sua base de dados e requisita ao TCD para criar os filtros;
20. o BB requisita o TCD para criar os filtros nos roteadores de borda;
21. o TCD faz uma chamada de sistema para criar o filtro nos nós DS;
22. caso aconteça algum erro na criação do filtro no TCD, o BB remove as informações deste filtro em sua base de dados e retorna um valor de erro para o Ambiente de Gerenciamento.

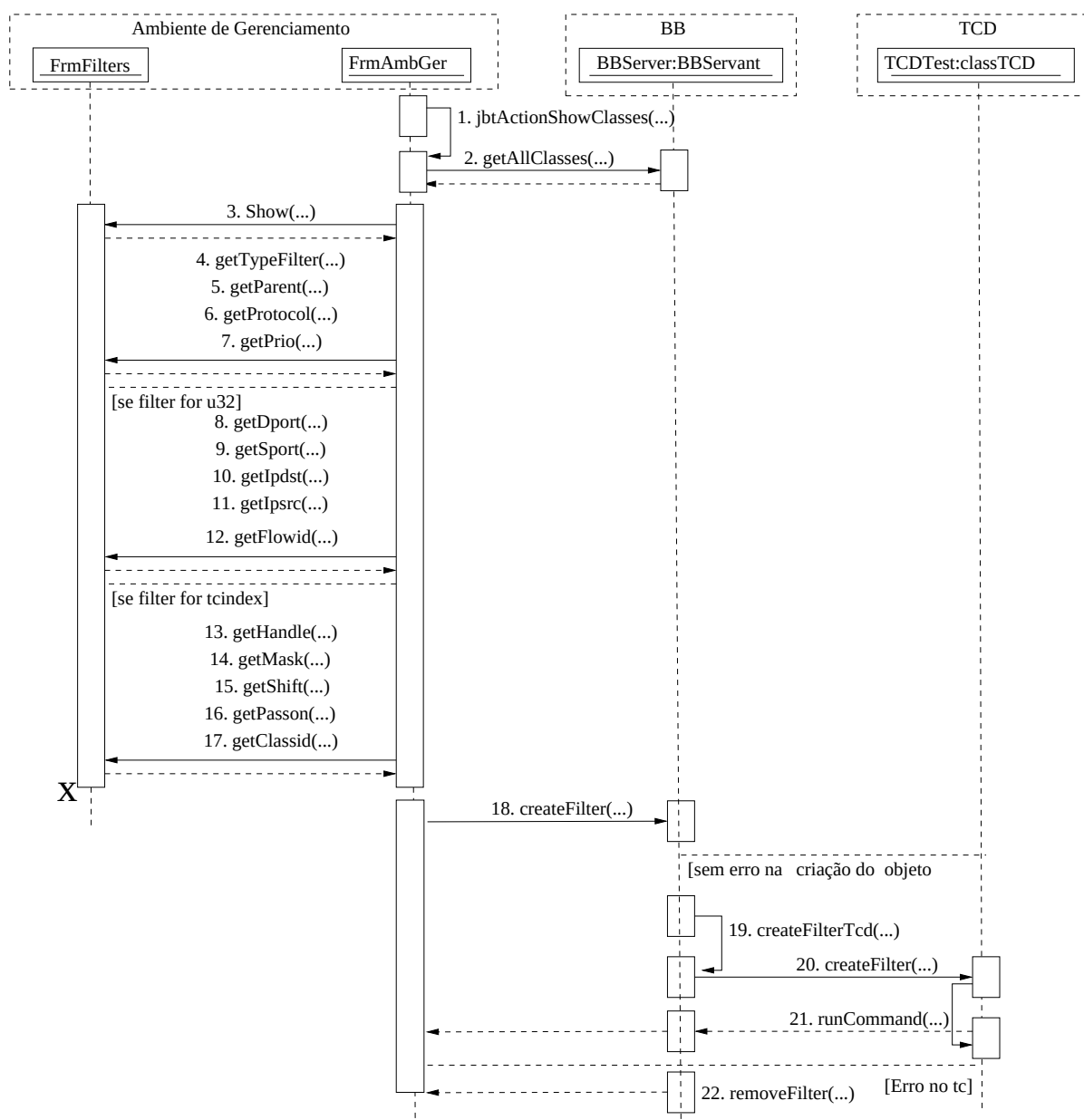


Fig. 3.21: Enquadramento de um Fluxo em uma Classe de Serviço DS.

3.3.5 Atualização do Uso de uma Classe de Serviço DS

A liberação de uma classe é realizada sempre que uma aplicação não necessita de garantias da rede. O diagrama Fig. 3.22 mostra a sequência de passos para realizar esta liberação.

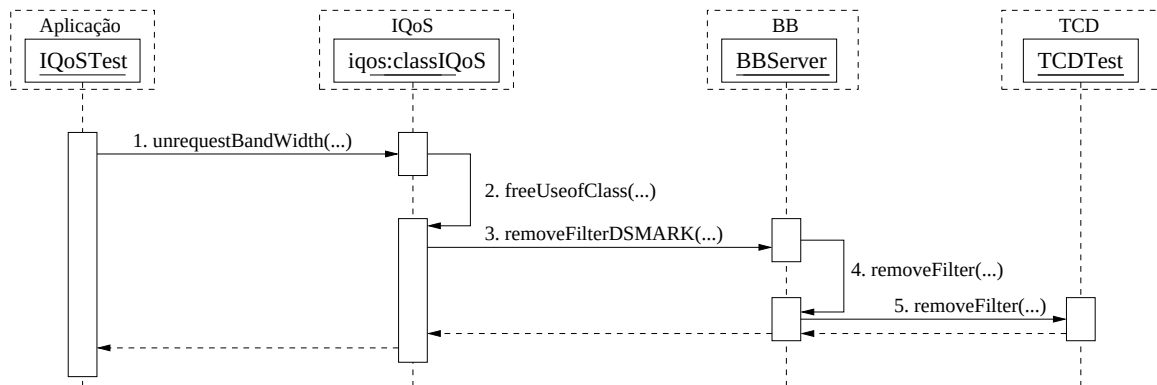


Fig. 3.22: Atualização do Uso de uma Classe de Serviço DS.

1. informar ao IQoS que não necessita mais de garantias;
2. o IQoS informa o BB para liberar a banda de transmissão utilizada pela aplicação;
3. o BB atualiza a tabela de contabilidade de uso das classes DS e requisita que o TCD remova o enquadramento dos fluxos na rede;
4. o TCD remove os filtros do controlador de tráfego nos roteadores do domínio DS.

Capítulo 4

Aspectos de Projeto e Tecnologias

Neste capítulo apresenta-se o projeto e as tecnologias utilizadas para implementar os componentes do modelo proposto. Este modelo é descrito em quatro seções principais: **i.** plataforma para objetos distribuídos, **ii.** sistema operacional com suporte ao controle de tráfego, **iii.** linguagem de programação para a implementação dos componentes; **iv.** ferramenta para gerar e analisar o tráfego.

4.1 Arquitetura CORBA

A Arquitetura CORBA (*Common Object Request Broker Architecture*) (Fig. 4.1) [34] do grupo OMG (*Object Management Group*) permite a interação entre objetos distribuídos em um sistema heterogêneo. Clientes podem executar operações de servidores sem se preocupar com detalhes como: localização do objeto invocado, linguagem de programação utilizada na implementação do servidor e diferenças existentes entre os ambientes dos objetos (p. ex., *hardware* e sistemas operacionais).

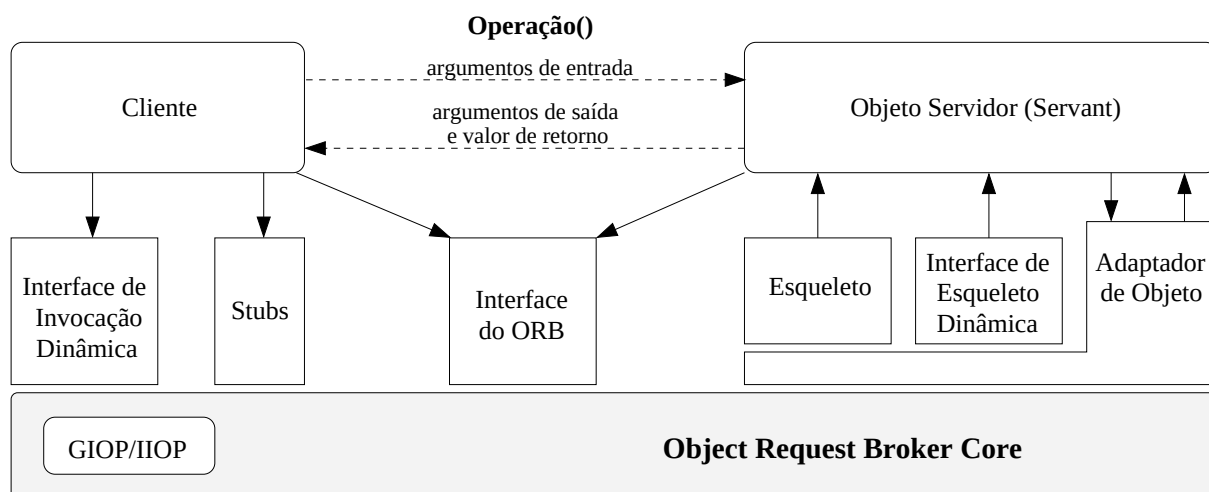


Fig. 4.1: Componentes da Arquitetura CORBA.

Uma descrição sucinta dos elementos que compõem a Arquitetura CORBA é apresentada a seguir:

- **Servant:** responsável pela execução das operações especificadas nas interfaces do objeto CORBA. A interface é definida por meio da linguagem declarativa IDL (*Interface Definition Language*) e os *servants* podem ser implementados em várias linguagens de programação;
- **Cliente:** para executar operações dos *servants*, o cliente deve obter uma referência do objeto que identifica unicamente o servidor. A forma como as operações dos *servants* são invocadas não depende da localização (local ou remota) do servidor em relação ao cliente;
- **Core ORB:** responsável por requisitar as operações para o servidor e, se for o caso, da resposta ao cliente. O ORB (*Object Request Broker*) pode ser implementado de várias formas como: processo servidor, recursos do sistema operacional, bibliotecas de funções. Para realizar a comunicação entre diferentes ORBs, utiliza-se o protocolo padrão IIOP (*Internet Inter-ORB Protocol*) [34], uma versão do GIOP (*General Inter-ORB Protocol*) que executa sobre o protocolo TCP;
- **Interface do ORB:** oferece algumas funções que podem ser diretamente utilizadas pelas aplicações (cliente ou servidores). Exemplo: conversões de referências de objetos para *strings* e vice-versa, criação de requisições dinâmicas;
- **Stubs e Esqueletos:** a função destes componentes é realizar conversão (*marshalling* e *unmarshalling*) dos dados enviados ou recebidos para uma representação canônica chamada de CDR (*Common Data Representation*). Normalmente são gerados de forma automática a partir da definição da interface do objeto CORBA;
- **DII (Dynamic Invocation Interface):** permite que um cliente invoque uma operação sem ter o conhecimento, em tempo de compilação, da interface (*stub*) do objeto servidor;
- **DSI (Dynamic Skeleton Interface):** função análoga à DII, utilizada por servidores, ou seja, um *servant* é capaz de atender requisições de clientes sem conhecer em tempo de compilação, a interface (esqueleto) em questão;
- **Adaptador de Objeto:** responsável pela associação entre o *servant* e o ORB. O adaptador é quem identifica para qual objeto deve ser enviada a requisição do cliente e qual operação deve ser executada. Outras funções dele estão relacionadas com o registro e a ativação dos *servants*.

O objetivo maior do consórcio OMG é definir uma arquitetura para a distribuição e interação de componentes de *software* orientado a objetos, em um ambiente heterogêneo. Os principais elementos dessa proposta são: uma arquitetura de referência denominada OMA (*Object Management Architecture*) e a especificação CORBA.

O principal componente da arquitetura OMA é o ORB, sendo ele o mecanismo de comunicação da arquitetura, funcionando como um "barramento" de *software* que fornece uma infra-estrutura, a qual permite a interação entre objetos, independente da plataforma e das técnicas utilizadas na implementação dos mesmos. A arquitetura do ORB, tratada pela especificação CORBA, foi definida para que diferentes implementações de ORB possam fornecer serviços e interfaces comuns.

Aplicações CORBA consistem de um conjunto de objetos que interagem através dos mecanismos de comunicação suportados pelo ORB e tratam do mapeamento de linguagens e protocolos para comunicação entre ORBs. As interações entre os componentes em CORBA são feitas sob os princípios de orientação a objetos e o paradigma cliente/servidor. Assim, no modelo de objetos CORBA tem acesso à referência de um objeto e invoca operações definidas na interface do objeto. O servidor CORBA corresponde à implementação de um objeto que recebe operações com seus parâmetros, executa-as e retorna o resultado (se for o caso).

Cabe ressaltar que os papéis de cliente ou servidor em CORBA correspondem apenas a uma divisão de papéis que ocorre durante a interação entre os objetos. Um objeto no ORB pode atuar tanto como cliente quanto como servidor, dependendo da ocasião. De fato, na terminologia CORBA, um cliente é visto simplesmente como o iniciador de uma requisição, não necessariamente como um objeto. O cliente é designado como um objeto se ele também assume o papel de servidor por alguma operação.

4.1.1 IDL - *Interface Definition Language*

Para que os clientes possam invocar operações nos objetos servidores, eles necessitam conhecer as operações (métodos) que o objeto servidor disponibiliza, bem como de que forma (parâmetros e tipos) essas operações devem ser invocadas, ou seja, eles precisam conhecer a interface do objeto servidor.

O Padrão CORBA define uma linguagem neutra, que permite especificar um objeto CORBA e suas interfaces, definindo-se, assim, o tipo do objeto, seus atributos, os métodos que suporta e os parâmetros desse método. É importante notar que cada objeto CORBA provê apenas uma única interface.

As interfaces IDL podem ser implementadas em qualquer linguagem para a qual o mapeamento IDL esteja disponível, como por exemplo, Java e C++. Assim, aplicações CORBA escritas em linguagens diferentes podem interoperar. A tarefa básica do projetista concentra-se em definir a especificação dos objetos e implementar o código dos métodos utilizando a linguagem que julgar mais apropriada.

4.1.2 Aspectos de Interoperabilidade

CORBA define o protocolo GIOP para a comunicação entre ORBs através da rede de comunicação, permitindo que haja interoperabilidade entre diferentes implementações de ORB. O protocolo GIOP foi projetado para trabalhar diretamente sobre qualquer protocolo de transporte orientado a conexão. Em particular, CORBA especifica o IIOP para a interconexão de ORBs através de uma rede TCP/IP.

Além de protocolo de interoperabilidade entre ORBs é utilizada também uma representação comum de dados - CDR (*Common Data Representation*) - que mapeia os tipos de dados definidos em IDL para um formato apropriado para transmissão através de uma rede, considerando aspectos inter-plataformas, como ordem de *bytes* e alinhamento de memória.

CORBA define também um formato para uma “referência de objetos interoperável” - IOR (*Interoperable Object Reference*). Ela é uma estrutura de dados que mantém informações necessárias para localizar um objeto sobre uma ou mais pilhas de protocolos, como acontece entre ORBs, por exemplo. Uma IOR contém dados que identificam, além do objeto, o domínio ao qual a referência está associada, ou seja, permite que um ORB consiga localizar a máquina, o servidor e o objeto a ser invocado.

Por fim, deve-se enfatizar que esses aspectos relacionados a interoperabilidade, como IIOP e CDR são de maneira geral, transparente para os desenvolvedores de aplicações. Além disso, aspectos como a localização dos objetos, a implementação de objetos e a ativação de um objeto (se necessário) antes de lhe entregar uma requisição também são transparentes às aplicações. Esses tipos de transparências, embora não exclusivas do CORBA, juntamente com outros aspectos contemplados pela especificação, como os “serviços CORBA” [34], muito contribuíram para tornar o CORBA uma alternativa bastante atraente para o desenvolvimento de aplicações cliente-servidor distribuídas em ambientes heterogêneos, incluindo a integração de sistemas legados.

4.1.3 Java e CORBA

A API Java IDL, presente na plataforma Java 2, permite a integração entre objetos Java e outros objetos remotos, eventualmente desenvolvidos em outras linguagens, através da arquitetura CORBA.

Os principais pacotes que compõem essa API são `org.omg.CORBA` e `org.omg.CosNaming`.

A partir da versão 1.3, é possível gerar interfaces IDL para classes Java usando o compilador `rmic` com a opção `-idl`. Outra opção, `-iiop`, indica que o protocolo de comunicação de CORBA, IIOP, será utilizado em *stubs* e *ties* (correspondentes aos *skeletons*) de RMI.

Uma vez obtida a interface IDL para um serviço, as classes auxiliares para acessar o objeto remoto que implementa o serviço são obtidas pela compilação, usando o aplicativo `idlj` (ou `idltojava` ou ainda `idl2java` em versões anteriores à Java 1.3). Além de classes para *stubs* e *skeletons*, são geradas classes auxiliares (*helpers* e *holders*) para permitir a comunicação entre objetos Java e outras linguagens.

Na plataforma Java há uma implementação para o serviço de nomes CORBA, oferecida pelo aplicativo `tnameserv`. Esse serviço está mapeado por `default` para a porta 900, podendo ser modificado pela opção `-ORBInitialPort`. A interação entre um ORB e um programa Java dá-se através de métodos da classe ORB. Para iniciar a referência ao ORB, utiliza-se o método estático `init()` dessa classe. Para obter uma referência para o serviço de nomes utiliza-se o método `resolve_initial_references()` tendo a *string NameService* como argumento [35, 36].

4.2 Tecnologia Web

A tecnologia Web é proposta no modelo de gerenciamento do domínio DiffServ. No escopo do modelo utilizado, tecnologia Web refere-se aos componentes e funcionalidades: cliente (*Browser*), protocolo HTTP (*HyperText Transfer Protocol*), HTML (*HyperText Markup Language*) e *applets* Java.

Sistemas baseados na tecnologia Web tem proliferado não só para novas aplicações, mas também como interface para sistemas já existentes. Inicialmente restrito à World Wide Web na *Internet*, o uso da tecnologia Web ultrapassou fronteiras e hoje está presente em sistemas de uso interno das empresas (*Intranets*) e em sistemas para diversos dispositivos (p. ex. celulares, PDAs, geladeiras, aparelho de som e etc.). A motivação para construir o componente Ambiente de Gerenciamento usando a tecnologia Web, é que desta maneira ele traz toda as vantagens desta tecnologia, entre elas, a capacidade de qualquer máquina com um *Browser* configurar o domínio DiffServ.

A tecnologia Java é utilizada para criar páginas da Web com conteúdo interativo e dinâmico, para desenvolver aplicativos corporativos de larga escala, para aprimorar a funcionalidade de servidores Web, fornecer aplicativos para dispositivos destinados ao consumidor final (como telefones celulares, *paggers* e PDAs) e assim por diante [37]. As vantagens da utilização da linguagem Java são: gratuita, orientada a objeto, independente de plataforma, integração com a Web e o suporte ao padrão CORBA.

4.2.1 Applets Java

Quando a interação entre o usuário e a página Web ultrapassa as capacidades do HTML, podemos utilizar de *plug-ins* para adicionar recursos extras aos *Browsers* (Navegadores). A linguagem Java preenche essa necessidade, pois com Java podemos escrever pequenas aplicações (*applets*) que são copiadas do servidor Web para para o *Browser*. Um *applet* é executado à medida que o usuário precisa dele e depois quando o usuário fecha o *Browser* ele é descartado automaticamente da memória [38, 39]. Esta tecnologia foi utilizada para a implementação do componente Ambiente de Gerenciamento.

4.3 Controle de Tráfego com IPROUTE2

O IPROUTE2 [30] é um pacote que contém dois comando: *ip* e o *tc*. O *ip* é utilizado para configurar o endereço IP dos *hosts*, tabelas de roteamento e etc. O comando *tc* é usado para criar

filtros, classes e para associar disciplinas de fila as interfaces de rede. A sintaxe do comando `tc` é:

```
tc [OPTION] OBJECT {COMMAND | help}
```

O campo `OBJECT` define o objeto que se pretende configurar. Este objeto pode ser:

- **qdisc**: disciplinas de fila;
- **class**: classes associadas as disciplinas;
- **filter**: filtros para classificação dos pacotes.

Se no lugar de um destes objetos for especificado o parâmetro **help**, informações de ajuda sobre a sintaxe básica do comando `tc` são apresentadas. As opções possíveis para o parâmetro **COMMAND** são específicas para cada tipo de objeto. Quanto ao parâmetro **OPTIONS**, as opções possíveis são:

- **stats**: para obter estatísticas sobre o objeto especificado;
- **details**: para aumentar o nível de detalhes na listagem dos objetos;
- **raw**: a saída de comando `tc` é mostrada de forma legível, porém ela é armazenada internamente em um formato codificado por razões de eficiência. Este parâmetro serve para mostrar a saída na forma armazenada internamente, em vez do formato legível;
- **version**: permite obter informação sobre a versão do comando `tc` em uso;
- **help**: mostra informação sobre a sintaxe básica do comando `tc`.

A Tab. 4.1, mostra as convenções utilizadas pelas ferramentas do IPROUTE2 que diferem do uso comum das grandezas de medidas usadas em redes.

Unidade	<i>bit(s)</i>
b	8
kb	1024 * 8
mb	1024 * 1024 * 8
kbit	1024
mbit	1024 * 1024
bps	8/segundo
kbps	1024 * 8/segundo
mbps	1024 * 1024 * 8/segundo
kbits	1024/segundo
mbits	1024 * 1024/segundo

Tab. 4.1: Convenções do Pacote IPROUTE2.

4.4 Controle de Tráfego no *Linux*

O Controle de Tráfego (Fig. 4.2) é o mecanismo que permite tratar os pacotes recebidos e/ou transmitidos em um roteador. A tarefa principal é decidir quais pacotes podem ser aceitos, e se aceitos, quais as vazões de entrada e saída na rede.

Na maioria das situações, o controle de tráfego consiste em uma simples fila que coleta os pacotes que entram e os encaminha rapidamente ao *hardware* (dispositivo de rede) tão rápido quanto ele admite. A disciplina de fila padrão no *Linux* é a `pfifo_fast`.

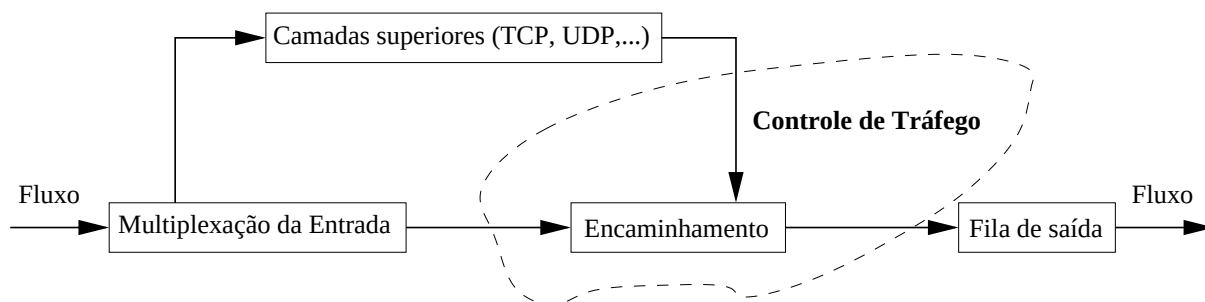


Fig. 4.2: Localização do Controle de Tráfego no *Linux*.

Redes baseadas em comutação de pacotes (as redes IP) diferem das redes baseadas em circuito por serem sem conexão. Redes baseadas em circuitos, tais como a de telefonia, sabem do estado da rede, assim as aplicações sabem a princípio qual os parâmetros do serviço que a rede poderá lhes oferecer. O controle de tráfego é uma possibilidade de simular o comportamento das redes baseadas em circuitos através do tratamento diferenciado dado aos pacotes. De fato, esta alternativa é utilizada para implantar o DiffServ em redes sem conexões.

As filas são locais de armazenamento temporário (algumas vezes chamado de *buffer*) que contém um número finito de itens esperando por uma ação ou serviço. Nas filas os pacotes esperam ser transmitidos à interface de rede. Em um modelo simples os pacotes são encaminhados na ordem em que eles chegam. Este modelo é conhecido como FIFO (*First In, First Out*) (Fig. 4.3) ou também é referenciado como FCFS (*First Come, First Serve*). Através desta disciplina de fila, nenhuma promessa pode ser feita pelo Controlador de Tráfego a não ser a ordem de saída dos pacotes.

As funções das filas são ampliadas quando elas são acopladas a outros mecanismos que podem atrasar, reordenar, priorizar pacotes em múltiplas filas. Através destes mecanismos, filas podem ter outras filas aumentando a complexidade da operação do escalonamento.

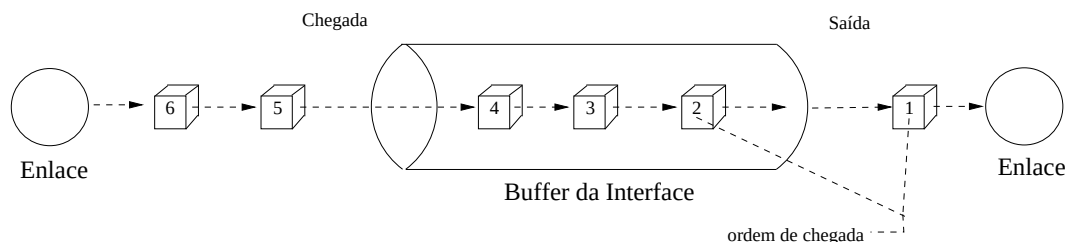


Fig. 4.3: Disciplina de Fila FIFO.

Um fluxo (Fig. 4.4) é um “diálogo” entre dois *hosts*. Uma sequência de pacotes que de alguma forma se diferencia dos demais é considerado um fluxo de pacote. O conceito de fluxo pode ser

estendido a vários cenários. Analisando a camada de transporte pelo protocolo TCP, uma sequência de datagramas que possuem os valores dos campos **sport** (porta de origem), **dport** (porta de destino), **ipsrc** (endereço IP da máquina de origem) e **ipdst** (endereço IP da máquina de destino) em comum pode ser considerado um fluxo de datagramas. O tráfego em uma rede pode ser constituído por um conjunto de fluxos. O mecanismo do controle de tráfego frequentemente separa os fluxos em classes de fluxos que podem ser agregados e tratados de forma distinta.

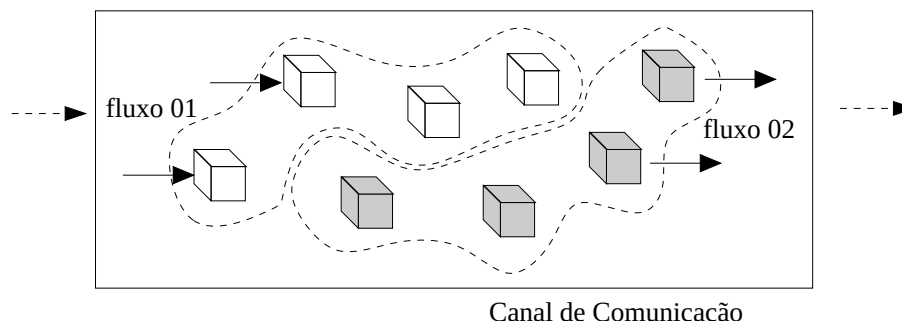


Fig. 4.4: Canal de Comunicação com Dois Fluxos Distintos.

4.4.1 Tratamento dos Pacotes

Kernels mais recentes do *Linux* já oferecem uma grande variedade de funções de controle de tráfego. Várias partes do *kernel* e vários programas utilizados no espaço do usuário têm sido desenvolvido por Alexey Kuznetsov [33]. A Fig. 4.5 mostra de uma forma simplificada o processo de recebimento de dados da rede. Os dados entrantes são examinados e passados para as camadas superiores ou depositados em filas de saídas para serem encaminhados à rede.

O encaminhamento inclui a seleção da interface de saída e o próximo nó da rede (*hop*). Os pacotes são enfileirados na fila associada à interface de saída. É nesse ponto que o controle de tráfego atua. O controle de tráfego pode decidir, entre outros, quais pacotes serão enfileirados, quais serão enviados primeiro e quais serão eliminados (por exemplo, se a taxa está excedendo o limite pré-fixado).

Cada dispositivo possui uma disciplina de fila associada a ele o qual controla como os pacotes serão tratados. Uma disciplina muito simples, consiste de apenas uma única fila, onde todos os pacotes são enfileirados na ordem que chegam, e os primeiros a serem enviados são o primeiros chegar.

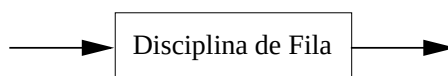


Fig. 4.5: Disciplina de Fila Simples sem a Utilização de Classes.

Disciplinas de fila (QDISC) mais elaboradas podem usar filtros para distinguir entre vários tipos de classes e processar cada classe de maneira específica. Como mostra a Fig. 4.6, algumas disciplinas possuem vários filtros que podem mapear para uma mesma classe, que por sua vez possui uma disciplina de fila associada, portanto disciplinas de fila e classes estão relacionadas.

Os pacotes são enfileirados de acordo com os seguintes passos. Primeiro a função de enfileiramento é chamada, ela percorre um filtro após o outro, até um onde a comparação seja satisfeita. Então um identificador de classe é achado, e o pacote é enfileirado na fila associada a essa classe. Pacotes que não

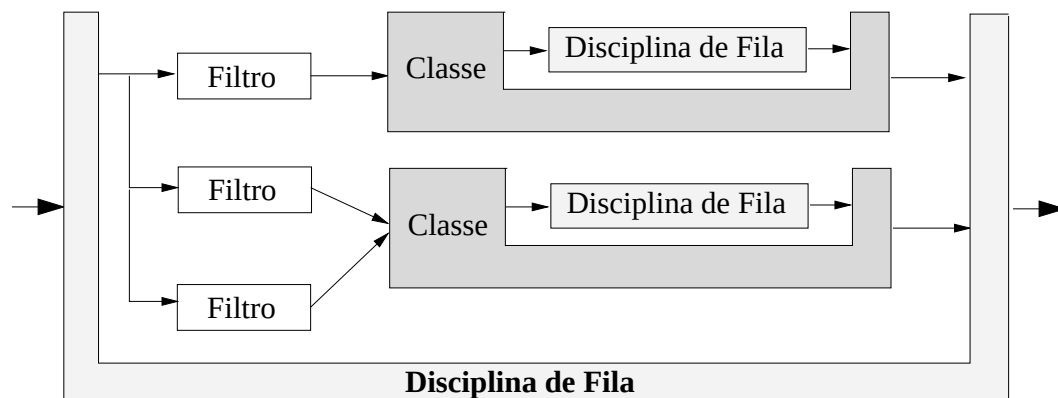


Fig. 4.6: Disciplina de Fila com a Utilização de Classes.

satisfazem nenhum filtro, receberão um identificador de classe padrão. Cada classe possui sua própria fila, várias classes podem compartilhar uma mesma fila.

4.4.2 Mecanismos de Modelagem com Tokens & Buckets

Fichas (*Token*) e Baldes (*Bucket*) (Fig. 4.7) são dois mecanismos de modelagem de tráfego. Um *token* pode representar uma quantidade de *bytes* ou simplesmente um pacote. O *Bucket* é um reservatório de *tokens*. Esses dois conceitos são utilizados para regular uma taxa de saída de pacotes por um canal de comunicação. A taxa de geração dos *tokens* corresponde à taxa de vazão dos pacotes. Esses conceitos são de grande importância e bastante utilizados pelos controladores de tráfego. A disciplina de fila HTB muito difundida entre os controladores de tráfegos é abordada neste capítulo.

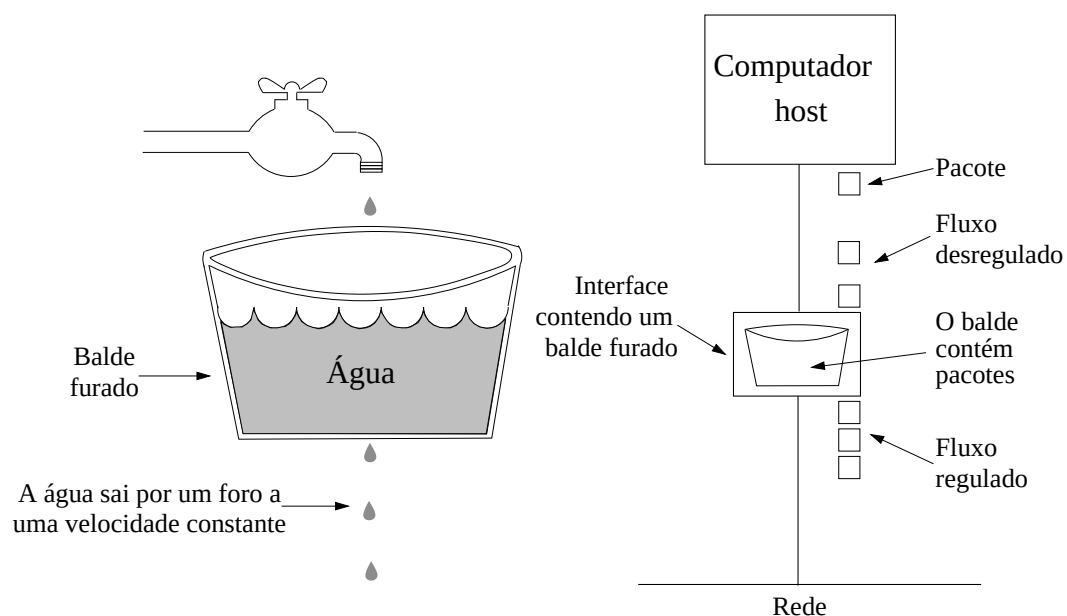


Fig. 4.7: Técnica Utilizada pela Fila HTB para Regular um Fluxo.

4.4.3 Disciplinas de Fila Utilizadas

As QDISCs constituem o ponto mais importante para que a Arquitetura DiffServ possa ser implementado. Através delas marcamos os pacotes, classificamos e alteramos o modo como são tratados.

Disciplina DSMARK

A QDISC DSMARK foi especialmente projetada por Werner Almesberger, Jamal Hadi Salim e Alexy Kuznetsov para satisfazer a especificação DiffServ. Sua função é realizar a marcação de pacotes [25, 40]. Essa disciplina não modela ou policia o tráfego, não prioriza, atrasa, reordena ou elimina pacotes, apenas marca o campo DSFIELD presente no cabeçalho do pacote IPv4.

As classes são enumeradas de 1 até n (Fig. 4.8), onde n é um parâmetro que define a quantidade de linhas da tabela DSMARK (Fig. 4.9). Essa tabela possui as colunas: índice, máscara e valor. Os valores dessas colunas após a seleção de uma linha, são utilizados para produzir um valor que será marcado no campo DSFIELD. A seleção é feita através de filtros, eles serão explicados na Seção 4.4.4.

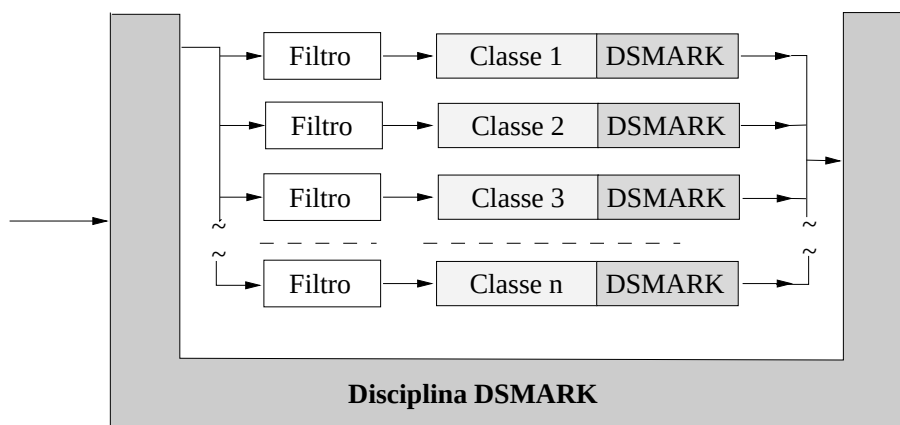


Fig. 4.8: Disciplina de Fila DSMARK.

Para criar uma nova disciplina usamos o seguinte comando:

```
tc add qdisc add dev eth0 handle <hd> root dsmark indices <id> [default_index
<did>] [set_tc_index]
```

O comando acima, configura a QDISC DSMARK na raiz da interface eth0 (primeira interface *Ethernet* conectada/instalada). O manipulador **hd** é o número utilizado para referenciar essa disciplina. O **id** é o número que indica a quantidade de classes a ser criada. O **did** é o número da classe padrão. Por fim o parâmetro **set_tc_index** é utilizado em conjunto com o filtro TCINDEX.

O comando abaixo, cria um QDISC DSMARK na raiz da interface eth0. O descritor da disciplina é o 1:0 que contém 32 elementos (classe 0 até a classe 31). A classe 1 é identificada como 1:1.

```
tc QDISC add dev eth0 handle 1:0 root dsmark indices 32
```

A Fig. 4.9 mostra o processo do cálculo do DSMARK para descobrir o valor do campo DSFIELD de um pacote qualquer. Através do filtro a classe 1:3 é selecionada. A tabela interna tem duas colunas nomeadas de máscara e valor. Os valores retornados são: máscara igual a 0x3 e valor igual a 0x68. Com esses dois valores, a QDISC DSMARK extrai o campo DSFIELD e aplica a seguinte operação:

$$Novo_DS = (Antigo_DS \& mascara) || valor \quad (4.1)$$

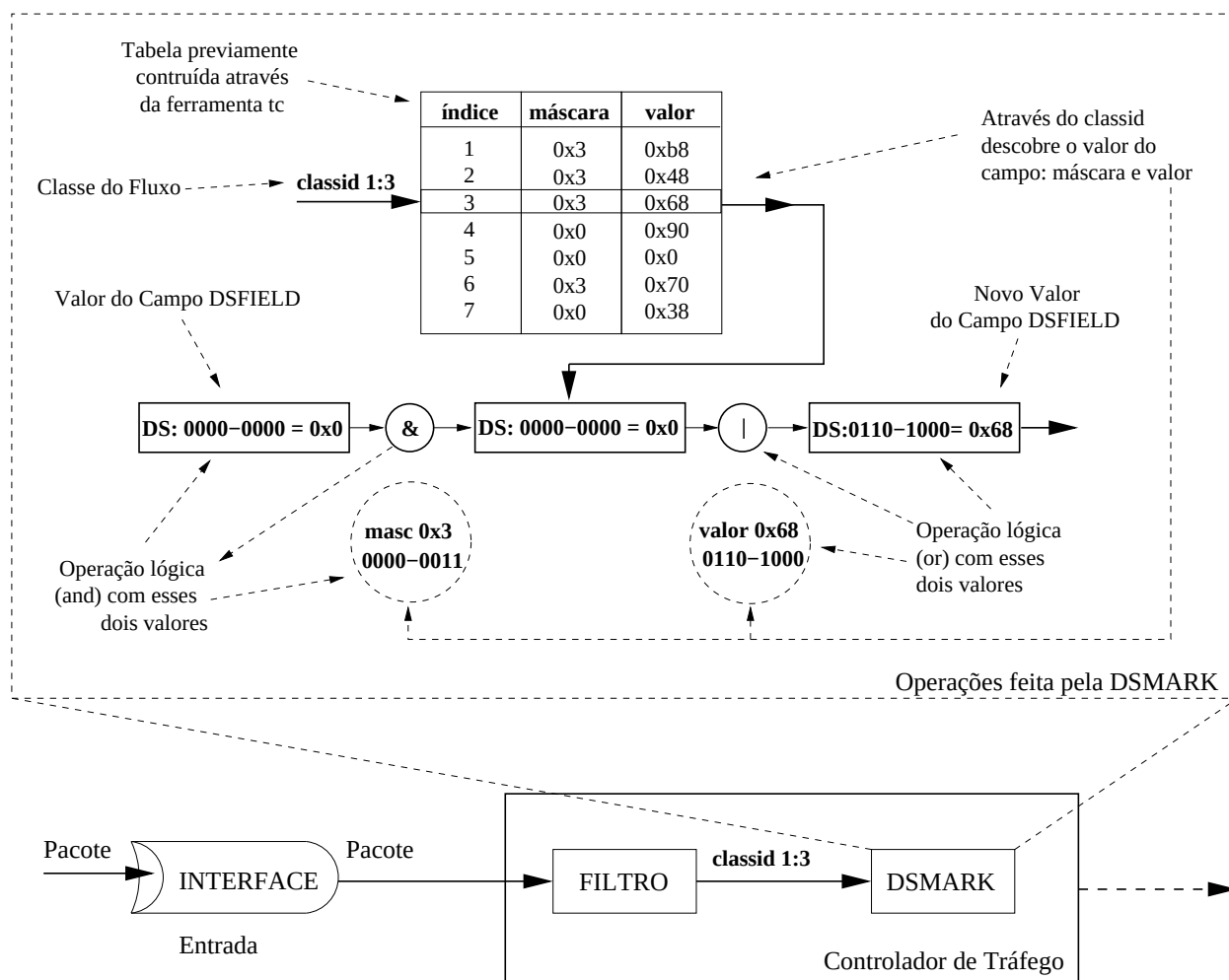


Fig. 4.9: Marcação através da Disciplina de Fila DSMARK.

O valor calculado é retornado ao campo DSFIELD. Nesse exemplo um pacote com DSFIELD igual a 0x0 (valor correspondente ao *Best Effort*) é selecionado. Após a operação lógica *and* com os dois números obteve o valor 0x0. E finalmente, a operação lógica *or* com os números 0x0 e 0x68, resultando o valor 0x68. Este processo mostra que um pacote entra com o valor do DSCP igual a 0x0 e segundo a tabela DSMARK, ele sai como o valor 0x68.

Observe que a máscara 0x3 preserva os dois *bits* mais a direita, eles podem ser utilizados em roteadores que estão habilitados a interpretar o ECN (*Explicit Congestion Network*).

As sequência de comandos abaixo mostra como criar a tabela utilizada no exemplo acima.

```
tc qdisc add dev eth0 handle 1:0 root dsmark indices 8
tc class change dev eth0 classid 1:1 dsmark mask 0x3 value 0xb8
tc class change dev eth0 classid 1:2 dsmark mask 0x3 value 0x48
tc class change dev eth0 classid 1:3 dsmark mask 0x3 value 0x68
```

```
tc class change dev eth0 classid 1:4 dsmark mask 0x3 value 0x90
tc class change dev eth0 classid 1:5 dsmark mask 0x3 value 0x0
tc class change dev eth0 classid 1:6 dsmark mask 0x3 value 0x70
tc class change dev eth0 classid 1:7 dsmark mask 0x3 value 0x38
```

A sequência abaixo mostra como criar os filtros utilizados no exemplo.

```
tc filter add dev eth0 parent 1:0 protocol ip prio1 u32 match ip src
  192.168.1.0/24 flowid 1:1
tc filter add dev eth0 parent 1:0 protocol ip prio1 u32 match ip src
  192.168.2.0/24 flowid 1:2
tc filter add dev eth0 parent 1:0 protocol ip prio1 u32 match ip src
  192.168.3.0/24 flowid 1:3
tc filter add dev eth0 parent 1:0 protocol ip prio1 u32 match ip src
  192.168.4.0/24 flowid 1:4
tc filter add dev eth0 parent 1:0 protocol ip prio1 u32 match ip src
  192.168.5.0/24 flowid 1:5
tc filter add dev eth0 parent 1:0 protocol ip prio1 u32 match ip src
  192.168.6.0/24 flowid 1:6
tc filter add dev eth0 parent 1:0 protocol ip prio1 u32 match ip src
  192.168.7.0/24 flowid 1:7
```

Na configuração apresentada, os pacotes destinados para a rede 192.168.3.0 são filtrados e classificados na classe 1:3, e depois de realizada as operações descritas na Fig. 4.9, ele sairá com o valor do campo DSFIELD com a 0x68.

O *Kernel* do *Linux* armazena em um *buffer* os pacotes entrantes. O código do *kernel* usa a estrutura nomeada por *skb* para acessar estes pacotes. Esta estrutura possui a propriedade chamada *tc_index*. O parâmetro *set_tc_index* obriga a cópia do campo DSFIELD para o campo *tc_index*. O campo *tc_index* também é utilizado em outras QDISC.

Disciplina GRED

A QDISC GRED (*Generalized Random Early Detection*) é uma derivação da QDISC RED (*Random Early Detection*) que foi criada por Werner Almesberger, Jamal Hadi Salim e Alexy Kuznetsov. Segundo Almesberger [41], a GRED foi projetada para atender às necessidades da Arquitetura DiffServ em relação ao comportamento AF que requer uma fila com múltiplas prioridades de descarte [42].

A QDISC RED se aproveita dos mecanismos de controle de congestionamento do protocolo de transporte TCP, o qual diminui a taxa de transmissão em vista da perdas de pacotes. O algoritmo RED tem como função a redução do tamanho médio das filas de espera. Esse tamanho é calculado com frequência. Se este exceder um valor mínimo determinado, é dada uma certa probabilidade de descarte aos pacotes que chegam. A medida que aumenta este excedente, a probabilidade de perdas de pacote também aumenta. Sempre que o comprimento médio ultrapassar um valor máximo todos os pacotes serão descartados. Esse mecanismo promove uma distribuição justa das perdas (tráfegos mais agressivos serão mais prejudicados) e visa evitar o efeito de sincronização global [25].

A GRED é utilizada em conjunto com a disciplina de fila DSMARK (Fig. 4.10), que através do seu parâmetro *set_tc_index*, copia o valor do campo DSFIELD para o campo *tc_index* da estrutura do *skb* mantida pelo *kernel* do *Linux*. Cada classe possui sua própria fila virtual. Os quatro *bits* mais a direita do *tc_index* são utilizados para selecionar a fila virtual.

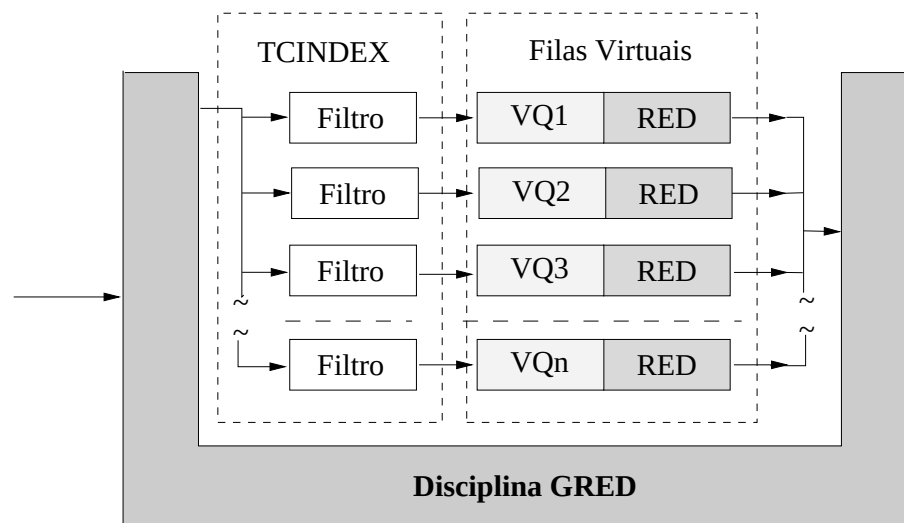


Fig. 4.10: Disciplina de Fila GRED.

Os seguintes parâmetros são utilizados para a configuração da GRED:

```
tc qdisc add dev eth0 root gred setup DPs <VQs> default <VQ> [prio]
```

Esse comando configura a disciplina GRED na raiz da interface `eth0`. O parâmetro `VQs` indica o número de filas virtuais a serem criadas e qual a fila virtual que será utilizada como padrão.

Se o parâmetro `prio` é fornecido, então cada fila virtual receberá uma prioridade. As prioridades variam de 1 até 16, cujo valor 1 (um) é a mais alta.

Após a criação das VQs o próximo passo é configurá-las, utilizando os seguintes parâmetros:

```
tc qdisc add change dev eth0 root gred limit <bytes> min <bytes> max <bytes>
burst <pacotes> avpkt <bytes> bandwidth <kbps> DP <vq> probability <number>
[prio <number>]
```

Segundo Balliache [43], os significados e as fórmulas para o cálculo desses parâmetros são:

- **limit**: é o tamanho máximo da fila em *bytes*;
- **min**: tamanho mínimo da fila em que os pacotes começam a receber a probabilidade de descarte. Um bom valor para o **min** é a metade do **max**;
- **max**: tamanho máximo da fila.
- **burst**: determina a média do tamanho da fila que é influenciada por rajadas de pacotes;
- **avpkt**: tamanho médio dos pacotes;
- **bandwidth**: tamanho da banda disponível;

- **probability**: probabilidade máxima de 0 até 1 para marcação. O valor sugerido é de 0.2;
- **DP**: identificador da fila virtual;
- **prio**: prioridade da fila, os valores são de 1 até 16.

Algumas fórmulas são sugeridas por Hubert [42] para a configuração dos parâmetros acima.

$$max = banda \cdot latencia \quad (4.2)$$

$$min = \frac{max}{2} \quad (4.3)$$

$$limite = 4 \cdot max \quad (4.4)$$

$$burst = \frac{(2 \cdot max + min)}{(3 \cdot avpkt)} \quad (4.5)$$

Disciplina HTB

Em situações onde os pacotes são tratados de uma mesma maneira não é necessário a utilização de classes, porém como no modelo DiffServ alguns pacotes possuem privilégios em relação a outros o uso de classe se faz necessário.

A disciplina CBQ (*Class Base Queue*) é utilizada por muito tempo para criar classes com a finalidade de modelar o tráfego. De acordo com o trabalho de Almesberger e Devik [42, 44], essa disciplina é complexa, difícil de configurar e de compreender. Nesse trabalho usamos a HTB (*Hierarchical Token Bucket*), que por sua vez é bem intuitiva e de fácil substituição da CBQ. Com a HTB [44, 42] é possível subdividir uma banda em várias sub-bandas e cada sub-banda derivada é associada a uma classe.

A Fig. 4.11 é um exemplo de HTB onde, a banda principal representada por 100% é dividida em duas sub-bandas, a sub-banda **A** usando 40% e a sub-banda **B** que usa os 60% da banda restante. A banda A é dividida em duas novas bandas, uma usando 30% destinada ao tráfego de páginas da *Internet* e outra de 10% reservada para tráfego de *e-mail*.

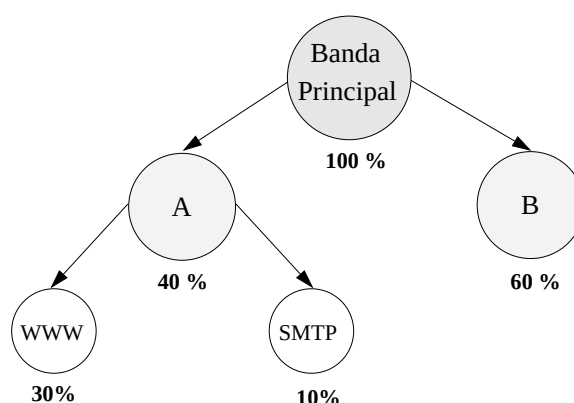


Fig. 4.11: Modelo Hierárquico de Classes HTB.

A seguinte sequência do `tc` apresenta como criar uma estrutura hierárquica de bandas como acima:

```
tc qdisc add dev eth0 root handle 1:0 htb default 12
tc class add dev eth0 parent 1: classid 1:1 htb rate 100kbps ceil 100kbps
tc class add dev eth0 parent 1: classid 1:2 htb rate 40kbps ceil 100kbps
tc class add dev eth0 parent 1: classid 1:10 htb rate 30kbps ceil 100kbps
tc class add dev eth0 parent 1: classid 1:11 htb rate 10kbps ceil 100kbps
tc class add dev eth0 parent 1: classid 1:12 htb rate 60kbps ceil 100kbps
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip
  dport 80 0xffff flowid 1:10
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip
  dport 25 0xffff flowid 1:11
```

A primeira linha do `tc` associa a QDISC HTB como raiz da interface `eth0`, isso permite que classes filhas sejam criadas, e define como classe padrão a classe 12. Se nenhuma procura pelos filtros forem satisfeita, então a classe 12, será selecionada.

A penúltima linha associa os pacotes referentes ao tráfego da *Internet*, cujo pacotes possuem o valor do campo da porta destino igual a 80, e os classificam na classe 1:10, e o segundo filtro identifica os pacotes referente ao tráfego de *e-mail*, que possuem o valor da porta destino igual a 25, e os classificam na classe 1:11. Se algum pacote não satisfizer os filtros, eles serão classificados na classe padrão 1:12.

Após criar as classes e os filtros é preciso associar uma QDISC as classes folhas (que ficam nas extremidades), os comandos abaixo descrevem como associar uma QDISC as classes folhas.

```
tc qdisc add dev eth0 parent 1:10 handle 20:0 pfifo limit 5
tc qdisc add dev eth0 parent 1:11 handle 30:0 pfifo limit 5
tc qdisc add dev eth0 parent 1:12 handle 40:0 pfifo limit 5
```

O Diagrama (Fig. 4.12) apresenta a configuração final das etapas realizadas acima:

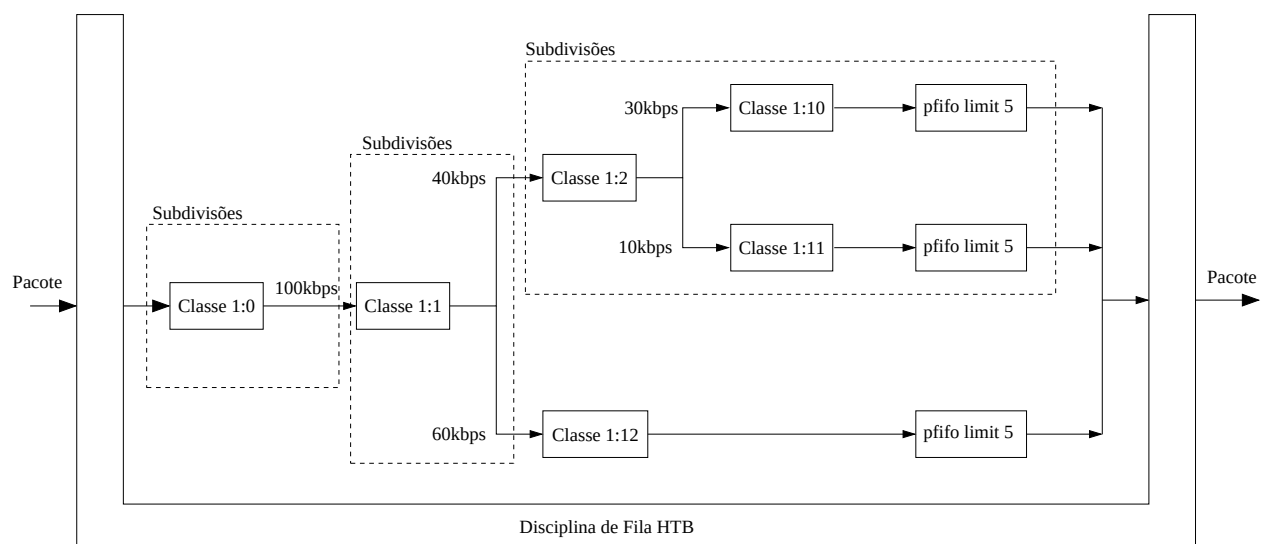


Fig. 4.12: Divisão de Banda de Transmissão através da Disciplina HTB.

4.4.4 Filtros e Classificadores

Destacamos nesta seção alguns classificadores presentes no *Linux*, tais como: U32 e TCINDEX.

Filtro U32

O U32 [42] é o principal classificador do *Linux*, pois ele permite a identificação de qualquer parte de um pacote IP (normalmente campos do cabeçalho). O U32 é bastante utilizado quando é necessário identificar IPs de origem e/ou destino, portas de origem/destino ou mesmo protocolo (ICMP, TCP, UDP, etc). Alguns dos principais parâmetros do U32 são mostrados na Tabela 4.2.

Passagem de Parâmetros para o Filtro U32	
Parâmetro	Significado
ip src IP/MASK	Casa com pacotes com origem em IP/MASK
ip dst IP/MASK	Casa com pacotes com destino IP/MASK
ip sport NN 0xffff	Casa com pacotes com porta de origem NN.
ip dport NN 0xffff	Casa com pacotes com porta de destino NN.
ip protocol NN 0xff	Casa com pacotes do tipo protocolo NN.
ip tos 0xNN 0xMM	Casa com os bits NN do campo TOS.
flowid X:Y	Os pacotes que casam com essa regra devem ser colocados na class X:Y

Tab. 4.2: Parâmetros do Filtro U32.

Filtro TCINDEX

O TCINDEX é um classificador utilizado em conjunto com outras QDISC. Ele foi especialmente projetado para implementar a Arquitetura DiffServ no *Linux*. O comportamento do TCINDEX (Fig. 4.13) é dependente do campo `skb->tc_index`. O `Skb` é um espaço criado para armazenar todos os pacotes que entram ou que são criados em uma máquina *Linux*. Este filtro é utilizado com disciplinas de fila que reconhecem o campo `tc_index` como a GRED, DSMARK e INGRESS.

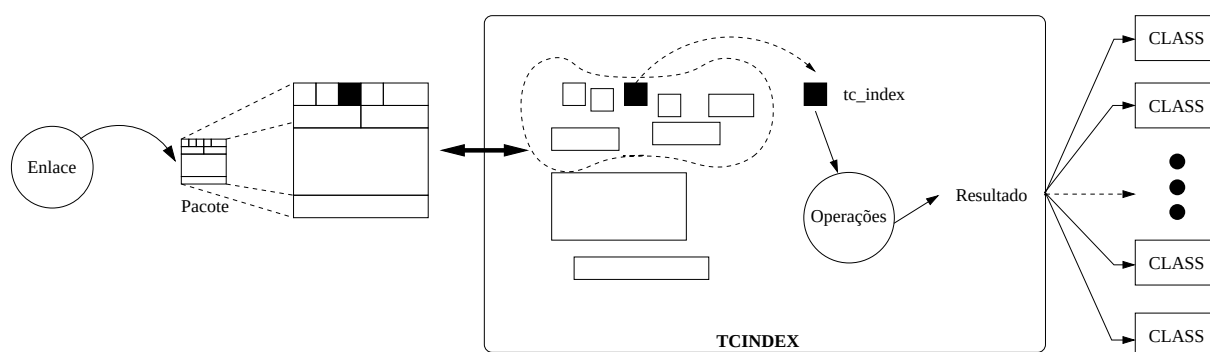


Fig. 4.13: Cópia do Campo DSFIELD para a Variável `tc_index`.

O valor do campo `tc_index` é lido pelo classificador, operações de *bitwise* (deslocamento de *bits*) são feitas e o resultado é utilizado para fazer uma busca nos filtros que compõem a estrutura do

TCINDEX. Após a conclusão da busca, um identificador de classe é retornado. O identificador é utilizado para selecionar uma classe e conseqüentemente uma fila. Um detalhe muito importante é que as operações realizadas não alteram o valor do campo DSFIELD, por isto a cópia do campo DSFIELD na estrutura `tc_index`. A sequência abaixo demonstra a utilização do filtro TCINDEX:

```
tc qdisc add dev eth0 handle 1:0 root dsmark indices 64 set_tc_index
tc filter add dev eth0 parent 1:0 protocol ip prio 1 tcindex mask 0xfc
    shift 2 pass_on
tc filter add dev eth0 parent 1:0 protocol ip prio 1 handle 10
    tcindex classid 1:111
tc filter add dev eth0 parent 1:0 protocol ip prio 1 handle 12
    tcindex classid 1:112
tc filter add dev eth0 parent 1:0 protocol ip prio 1 handle 14
    tcindex classid 1:113
```

A primeira linha associa a disciplina DSMARK como raiz da interface `eth0`, e faz que o campo DSFIELD seja copiado para o campo `tc_index`. Na segunda linha, a operação lógica *and* é realizada com o `tc_index` e o valor `0xFF`, o resultado é deslocado dois *bits* para a direita. O valor encontrado é passado para os filtros abaixo.

Se o valor resultante da operação acima for 10, a classe 1:111 será retornada, se for 11 a classe 1:112 será retornada e, por fim, se o valor for 14 a classe 1:113 será retornada. Por exemplo, se um pacote com o DSFIELD igual a `0x30`, o valor do `tc_index` será `0x30`. O classificador TCINDEX é chamado e a seguinte operação é realizada:

```
1. (skb->tc_index & mask) >> shift
2. (0x30 and 0xFC) >> 2
3. (00110000 and 11111100) >> 2
4. 00110000 >> 2
5. 00001100
Resultado: 0xC
```

O resultado é `0xC` significa 12 em decimal, portanto, o filtro com *handle* igual a 12 é selecionado e a classe 1:112 é retornada para a função de enfileiramento. O valor 112 é copiado para o `tc_index` da seguinte forma: `000100010010`.

4.5 Gerando Tráfego com o RUDE e CRUDE

As ferramentas RUDE (*Real Time UDP Data Emite*r) e CRUDE (*Collector Real Time UDP Data Emite*r) pertencem a um mesmo pacote. Elas são utilizadas para gerar tráfego na rede de acordo com um arquivo de *script*. Nesse arquivo são passados parâmetros como o tempo de duração, início de cada fluxo, porta origem, porta destino, endereço destino, tamanho dos pacotes, taxa de geração dos pacotes e a definição do campo TOS do protocolo IPv4. A forma de utilizar o RUDE é :

```
rude -s file [ -h ] [ -v ] [ -P 0-90 ]
```

onde:

- **-s file:** *script* que possui os fluxos a serem criados, bem como o seus comportamentos;
- **-h:** imprime um pequena ajuda (*help*);
- **-v:** imprime informações sobre a versão;
- **-P 0-90:** prioridade de execução.

O *script* é um arquivo de configuração que define a quantidade dos fluxos e o comportamento dos fluxos. Três comandos são essenciais: **START**, **ON** e **OFF**. Caso exista algum erro no arquivo, o programa é encerrado e nenhum fluxo é gerado. Abaixo é apresentado um exemplo de *script*:

```
START NOW
1000 0030 3002 192.168.0.1:10001 CONSTANT 200 250
3000 0030 MODIFY CONSTANT 400 500
4000 0030 MODIFY CONSTANT 1000 1000
TOS 0030 0x10
9000 0030 OFF
```

A primeira linha configura o momento em que os fluxos são criados, no lugar do **NOW** pode passar um horário específico (p.ex: **START 16:40:01**). Na segunda linha, o fluxo identificado por 30 é gerado após 1 segundo depois do comando **START**, esse fluxo possui as seguintes características: porta UDP de origem 3002, IP destino 192.168.0.1, porta UDP destino 10001. O fluxo é criado com uma taxa de 200 pacotes por segundo, onde cada pacote tem o tamanho de 250 *bytes*. A terceira linha modifica a taxa para uma vazão de 200 *kbytes/sec* e a quarta linha altera para 1000 *kbytes/sec*. A penúltima linha marca o campo TOS com o valor 0x10, e a última linha finaliza o fluxo 0030 com o parâmetro **OFF**.

O CRUDE é uma ferramenta para receber o tráfego gerado pela ferramenta RUDE, ela armazena informações estatísticas sobre os fluxos em um arquivo de *log*. A forma de utilizar o CRUDE é:

```
crude [-h ] [-v] [-d file] [-P 0-90 ] [-p port] [-i ip_addr] [-l file]
      [-s flow_id_list] [-n number]
```

Onde:

- **-h:** imprime um pequena ajuda (*help*);
- **-v:** imprime informações sobre a versão;
- **-d:** decodifica o arquivo gerado pela opção **-l**;
- **-P:** prioridade de execução;
- **-p:** configura a porta UDP de recebimentos de dados. Por padrão usa a porta 10001;
- **-i:** configura a interface de rede utilizada para receber os dados, para isso um número IP dever ser passado, caso nenhum valor seja passado o programa irá escutar todas as interfaces;
- **-l:** cria um arquivo utilizado para armazenar *log* em formato binário;
- **-s:** imprime dados estatísticos sobre um fluxo. Os dados são: número de pacotes recebidos, número de pacotes recebidos fora da sequência, número estimado de pacotes perdidos, total de *bytes* recebidos, *jitter* absoluto máximo e vazão total;
- **-n:** quantidade de pacotes para capturar.

4.6 Avaliação dos Parâmetros de QoS com TCPDump

O `tcpdump` imprime na tela o cabeçalho dos pacotes TCP, UDP e IP. Ele permite filtrar pacotes e salvar seus cabeçalhos em um arquivo para uma futura análise. A sintaxe do comando é:

```
tcpdump [ -adeflnNOpqRStuvxX ] [ -c count ] [ -C file_size ]  
        [ -F file ] [ -i interface ] [ -m module ]  
        [ -r file ] [ -s snaplen ] [ -T type ] [ -w file ]  
        [ -E algo:secret ] [ expression ]
```

O `tcpdump` possui várias opções e várias delas têm um alto grau de complexidade. Utiliza-se o TCPDump para verificar se o campo TOS do protocolo IP está marcado corretamente e armazenar os fluxos através da opção `-w` em um arquivo para cálculos estatísticos da vazão de cada fluxo.

4.7 Considerações Finais

Apresentou-se neste capítulo o modelo de tecnologia adotada para a implantação da Arquitetura DiffServ em um ambiente composto por máquinas que utilizam o sistema operacional *Linux*. A plataforma *Linux* foi adotada por disponibilizar recursos inclusos no seu próprio *Kernel* e através de programas que podem ser instalados. Tanto o *Linux* como os programas descritos neste capítulo são *softwares* com código fonte aberto (Open Software) [45].

Este modelo de tecnologias possibilitou o desenvolvimento dos componentes bem como a adição dos mesmos ao DiffServ (Capítulo 3). O próximo capítulo aborda os cenários de testes que possibilitaram a avaliação da Arquitetura DiffServ e o seu comportamento com a inclusão dos novos componentes.

Capítulo 5

Descrição dos Experimentos e Resultados

Nos capítulos anteriores apresentou-se as deficiências da Arquitetura DiffServ quanto ao controle de admissão e de gerencia das Classes DS. Nesse capítulo, apresenta-se um conjunto de testes para avaliar os componentes adicionados à Arquitetura DiffServ. Os testes são feitos em dois ambientes: o primeiro, formado por três microcomputadores e duas sub-redes (Fig. 5.1), enquanto no segundo, utilizou-se cinco computadores e cinco sub-redes (Fig. 5.11). Os componentes e as funcionalidades de cada máquina utilizada no primeiro ambiente são descritas abaixo:

- *Black* acomoda os componentes TCD (*Traffic Control Daemon*) e o IQoS (*Interceptor IQoS*). Tem a função de gerar fluxos e marcar os pacotes;
- *Brutos* tem a função de núcleo do domínio DiffServ, tais funcionalidades são desempenhadas pelo TCD, BB (*Bandwidth Broker*) e pelo Ambiente de Gerenciamento. Esta máquina tem duas interfaces de rede e se comporta como *gateway* da rede;
- *Inspirion* é o destino dos fluxos gerados pela máquina *Black*. Possui a função de gerar dados estatísticos sobre as vazões dos fluxos entrantes.

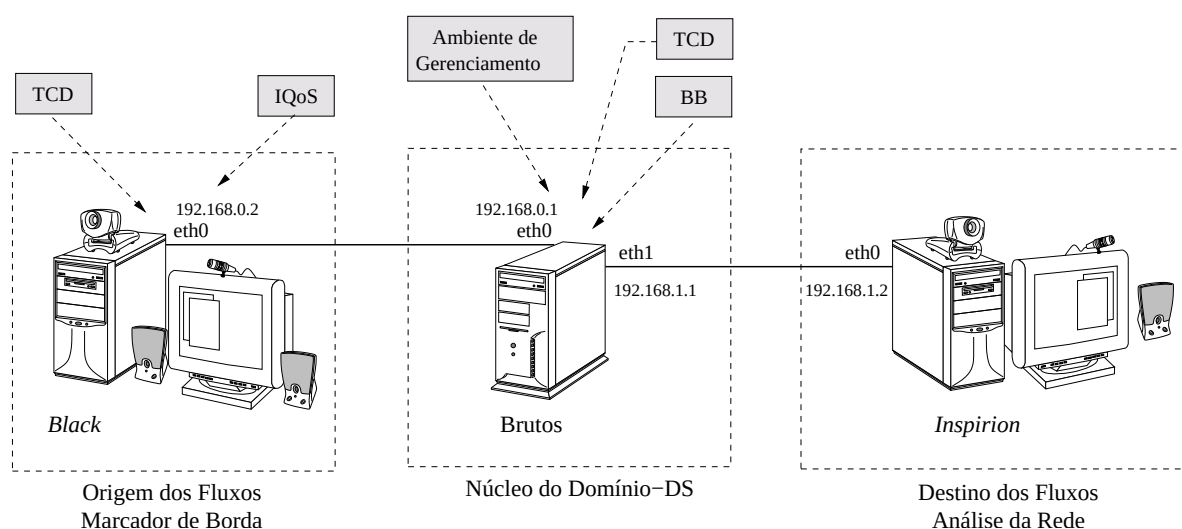


Fig. 5.1: Localização dos Componentes no Ambiente de Teste.

Características Principais			
Máquina	Processador	Memória	Rede (Banda)
<i>Black</i>	Pentium 160Mhz	48MB	100Mbps
Brutos	AtlhonXP 1.8 - 1,6Ghz	256MB	100Mbps
<i>Inspirion</i>	Pentium III 600Mhz	128MB	100Mbps

Tab. 5.1: Características das Máquinas Utilizadas nos Experimentos.

As principais características de *hardware* destes equipamentos estão disponíveis na Tab. 5.1.

Os testes são organizados em três fases para uma correta avaliação de todos os componentes do modelo tecnológico e da arquitetura de implementação. A primeira fase procurou certificar se as ferramentas escolhidas para gerar o fluxo e computar as estatísticas atendem às necessidades impostas pelo problema de controlar o QoS (*Quality of Service*) do domínio DiffServ; a segunda fase focou os esforços em implementar a arquitetura DiffServ e avaliar os seus resultados efetivos na rede; e a terceira e última fase teve por objetivo avaliar se o BB realmente contempla o controle das classes DS e se o seu estado retorna ao esperado em determinadas situações da rede.

5.1 Avaliação das Ferramentas

No primeiro teste, vários fluxos são gerados na máquina *Black* e direcionados para a máquina *Inspirion* (Tab. 5.2). O objetivo é verificar se o comportamento dos fluxos tem correspondência com os parâmetros utilizados nas instanciações. Todos os pacotes criados tem a mesma prioridade e o Domínio DiffServ proposto ainda não foi instanciado. Os itens a seguir, descrevem o significado de cada coluna que compõe a Tab. 5.2:

- etapa: composta por apenas um único fluxo com cada linha representando um teste realizado – o período de cada teste é de 10 (dez) segundos;
- tamanho do pacote: tem o mesmo valor (permanece do início ao fim de cada etapa);
- pacotes/seg: quantidade de pacotes por segundo gerado pela ferramenta RUDE — multiplicando esta coluna pela coluna *Tam. do Pacote* encontra-se a vazão teórica que a rede é submetida;
- vazão esperada: obtida através da multiplicação da coluna *Tam. do Pacote* por *Pacotes/seg* — o resultado é o valor teórico, utilizado como parâmetro para avaliar a ferramenta;
- vazão real: vazão real obtida através das ferramentas CRUDE e TCPCDump (instanciadas na máquina *Inspirion*);
- perdas: quantidade de pacotes perdidos em cada etapa.

Neste exemplo e em particular na etapa 05 (cinco) houve perdas de pacotes, este ponto do teste merece muita atenção. Como dito no parágrafo anterior, o domínio DiffServ não está configurado, e não existe nenhum policiamento previamente estabelecido. As perdas dos pacotes neste momento podem ter como possíveis justificativas: o limite máximo de transmissão do enlace ou a capacidade máxima de processamento dos pacotes alcançada pelo roteador. Nos testes futuros, que abordam o comportamento

Etapa	Tam. do Pacote (bytes - B)	Nro. Pacotes por Segundo	Vazão Esperada (Bps)	Vazão Obtida (Bps)	Nro. Pacotes Perdidos	Perdas (%)
01	200	250	50.000	50.029	0	(0%)
02	300	300	90.000	90.048	0	(0%)
03	400	400	160.000	160.054	0	(0%)
04	1000	1000	1.000.000	1.000.200	0	(0%)
05	1000	6000	6.000.000	5.989.830	19	(0,30%)
06	1000	6000	6.000.000	5.985.560	26	(0,45%)
07	1000	9000	9.000.000	8.964.110	42	(0,46%)
08	1000	9000	9.000.000	8.938.720	70	(0,78%)
09	1000	10000	10.000.000	9.903.060	99	(0,99%)
10	1000	10000	10.000.000	9.930.850	71	(0,71%)
11	1000	10000	10.000.000	10.001.800	0	(0%)
12	1000	10000	10.000.000	9.971850	31	(0,31%)
13	1000	12000	12.000.000	4.278.560	6386	(53,21%)
14	1000	12000	12.000.000	10.060.600	1514	(12,84%)
15	1000	13000	13.000.000	7.097.160	4469	(34,37%)
16	1000	13000	13.000.000	6.461.820	4964	(38,18%)

Tab. 5.2: Avaliação da Banda de Transmissão no Cenário de Teste.

no domínio DiffServ dos fluxos gerados na rede, é necessário certificar-se que a eliminação dos pacotes se dê pelo Modelador e não por outro elemento, por isto a necessidade deste teste.

Como mostrado na Tab. 5.1 todas as interfaces são de 100Mbps (*Bits* por segundo), ou seja, 12,5MBps (*Mega Bytes* por segundo). As ferramentas RUDE e CRUDE trabalham com a grandeza de Bps, por isso a sua utilização. A vazão esperada entre as máquinas é teoricamente de 12,5 MBps.

Para ilustrar esta fase, as linhas abaixo descrevem a saída da ferramenta CRUDE. A última linha mostra o cálculo feito pelo próprio programa sobre a vazão (*Throughput*) da Etapa 01.

```
crude -s 0030
crude version 0.62, Copyright (C) 1999 Juha Laine and Sampo Saaristo
crude comes with ABSOLUTELY NO WARRANTY!
This is free software, and you are welcome to redistribute it
under GNU GENERAL PUBLIC LICENSE Version 2.
Runtime statistics results:
-----
Flow_ID=30
Packets: received=2001   out-of-seq=0   lost(est)=0
Total bytes received=500250
Sequence numbers: first=0   last=2000
Delay: average = 9336308.532936   jitter=0.000021   seconds
Absolute maximum jitter=0.002222   seconds
Throughput=50029 Bps (from first to last packet received)
```

Após a sequência de testes, as seguintes conclusões podem ser extraídas. A primeira, e talvez a mais importante neste momento, é que a ferramenta RUDE respeita as configurações, e portanto oferece credibilidade na realização de testes para avaliar a arquitetura proposta.

A segunda conclusão é sobre as perdas de pacotes. Na Etapa 05, houve 19 pacotes perdidos. Neste ponto, os motivos das perdas devem ser conhecidos. Uma possível justificativa é a do enlace ter atingido o seu limite de transmissão. Nesta etapa o enlace é submetido a um tráfego de 6MBps, este valor significa o uso de apenas 48% da banda total teoria do enlace, e assim, descarta-se a possibilidade de sobrecarga na rede.

Nas etapas 06, 07, 08, 09 e 10 as perdas de pacotes sugerem que a rede foi submetida a um tráfego acima de sua capacidade de entrega de pacotes sem que houvesse perdas. Na etapa 11 criou-se um tráfego de 10MBps sem perdas de pacotes e, assim pode-se concluir que o enlace não estava no seu limite. A partir da vazão de 10MBps a quantidade de pacotes perdidos é consideravelmente mais alta, o percentual de pacotes perdidos passa de 0,31% para 53.21%.

Com base nos resultados, quanto à perda de pacotes deve-se desconsiderar perdas por falha de roteamento uma vez que se trata de uma topologia ponto a ponto. Então não ocorrem perdas por rotas erradas ou por falha em *hub* ou *switch*. Estes dispositivos, utilizados para conectar fisicamente as máquinas não foram contemplados em nosso ambiente, como mostra a Fig. 5.1. A conclusão é que as perdas estão relacionadas com a sobrecarga na máquina Brutos e com o processamento da transmissão de pacotes de uma interface de rede para outra. A máquina Brutos é uma máquina padrão PC (*Personal Computer*) e não um roteador projetado para está tarefa.

Para validar a conclusão sobre perdas de pacotes repetiu-se os testes referentes às etapas em que se detectou perdas de pacotes, porém alterando a transferência de dados da máquina *Black* para a máquina Brutos. Como as máquinas estão ligadas ponto a ponto os pacotes, após a criação, são transferidos diretamente para a interface de rede, não ocorre nesse caso transferência de pacotes de uma interface de rede para outra interface numa mesmas máquina.

Segundo Teste					
Etapa	Tam. do Pacote (bytes)	Pacotes/seg	Vazão Esperada (Bps)	Vazão (Bps)	Perdas
01	1000	6000	6.000.000	5.999620	0
02	1000	9000	9.000.000	9.000520	0
03	1000	12000	12.000.000	11.987500	7
04	1000	13000	13.000.000	12.027.400	741

Tab. 5.3: Avaliação da Banda de Transmissão e Perdas de Pacotes com a Ferramenta RUDE.

No primeiro teste nas etapas 06, 09, 13 e 16 houve perdas de, 26, 99, 6386 e 4964 pacotes. No segundo teste houve um queda significativa na perda de pacotes, em quatro teste com o perfil do fluxo igual ao do primeiro, a perdas foram 0, 0, 7 e 741. Confirma-se que as perdas nas etapas do primeiro teste são ocasionados pela transferência de pacotes de uma interface para outra. Este processo requer tempo de processamento e ocasiona um atraso para retirar um pacote de uma fila. O processo de retirar um pacote de uma fila e a constante chegada de pacotes neste período provoca a falta de espaço na fila de chegada, uma vez a fila cheia, os novos pacotes têm que ser eliminados por falta de espaço.

A criação de pacotes em determinada frequência não é exatamente como configurado. Por exemplo, no segundo teste (Tab. 5.3) e na segunda etapa a vazão real é um pouco maior do que a esperada. Segundo a análise do CRUDE em dez segundo são recebidos 1001 pacotes, esperava-se apenas 1000. A diferença entre os pacotes recebidos e esperados influenciou no resultado da média, que significa um aumento de 0,0057% a mais, esta diferença é insignificante.

A ferramenta RUDE e CRUDE segundo os testes satisfazem o processo de gerar e analisar o tráfego na rede. As perdas de pacotes aumentaram quando aumenta a quantidade de pacotes por segundo. Não é objetivo do trabalho avaliar os motivos das perdas, mesmo assim apresentam-se algumas justificativas para alguns casos. O objetivo principal é em determinar em qual taxa a rede é submetida sem que haja perdas de pacotes. Os fluxos a uma taxa de 10MBps, neste nosso ambiente de teste, permitem uma avaliação futura do domínio DiffServ com um alto grau de confiança no aspecto do policiamento.

Apesar da existência de vários parâmetros de QoS da camada de rede, nos testes o foco é na garantia de banda de transmissão à aplicação. Na próxima seção utiliza-se dos componentes desenvolvidos para configurar as classes DS no domínio, o foco dos testes é analisar como a rede se comporta.

5.2 Resposta DiffServ no Suporte à QoS

Esta fase tem a finalidade de avaliar o domínio DiffServ quanto ao privilégio dos fluxos. Rastreando a vazão de cada fluxo avalia-se quais serviços condizem com o esperado. Os resultados obtidos são ilustrados por gráficos da vazão do fluxo no tempo.

5.2.1 Comportamento dos Fluxos sem o DiffServ

O primeiro teste tem a função de estudar o comportamento da rede sem a implantação da Arquitetura DiffServ. O teste consiste de dois fluxos da máquina *Black* à *Inspiration* (Fig. 5.2). A duração deles é de 40 segundos. O primeiro fluxo em todo o período submete à rede uma taxa de 1MBps. O segundo, possui sua taxa alterada da seguinte maneira: inicialmente com uma taxa de 100KBps; depois de 15 segundos passa para 500KBps e após a mais 10 segundos é alterada para uma taxa de 1MBps.

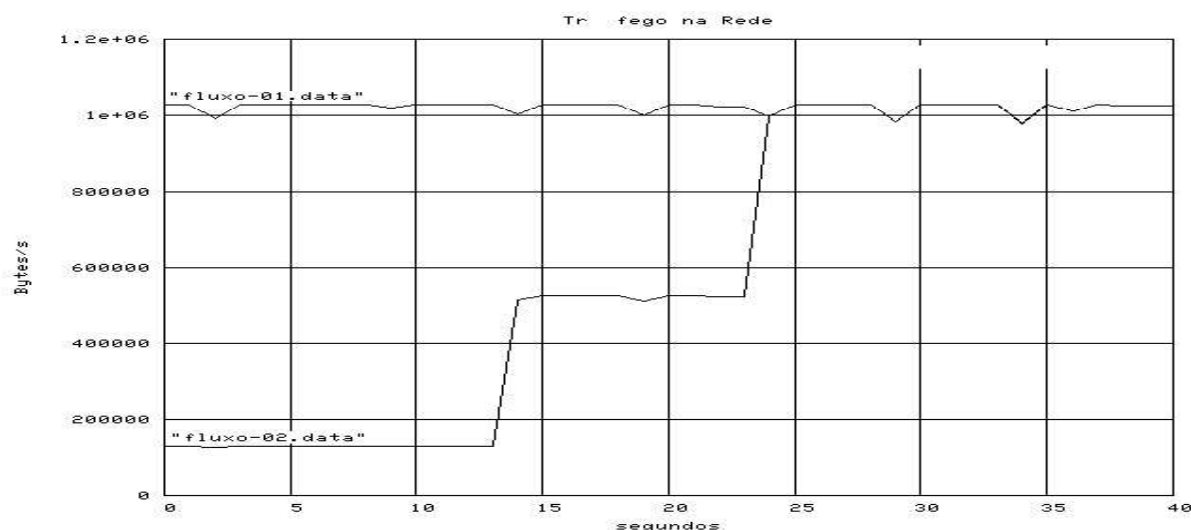


Fig. 5.2: Avaliação da Geração de Fluxo através da Ferramenta RUDE.

A Fig. 5.2 retrata o comportamento da rede do início ao fim. O fluxo intitulado “*fluxo-01*” manteve uma vazão de 1MBps e o fluxo intitulado “*fluxo-02*”, no intervalo de 0 a 15 manteve-se fixo em 200KBps. Um pouco antes dos 15 segundos a vazão do “*fluxo-02*” é antecipada em dois segundos, este pequeno adiantamento não implica em nenhum problema para avaliar as classes de serviços que são criadas no próximo teste.

Observa-se que os fluxos não associados a uma classe DS comportaram-se como esperado. Um fluxo mantendo-se constante no período dos quarenta segundos e o outro crescendo em valores estabelecidos até que compartilhem igualmente toda a banda de transmissão.

5.2.2 Comportamento dos Fluxos com o DiffServ

Neste segundo teste (Fig. 5.3), cria-se uma classe que tem uma banda total de 1MBps. Toda a banda é utilizada por apenas dois fluxos. O serviço de melhor esforço é associado ao fluxo intitulado como “*fluxo-be*”. O segundo fluxo, o “*fluxo-af*”, recebe um tratamento diferenciado, ele deve usar a banda total, quando necessário, em detrimento da banda utilizada pelo outro. As características dos fluxos são iguais ao do exemplo anterior, por isso a ferramenta RUDE utiliza o mesmo *script*.

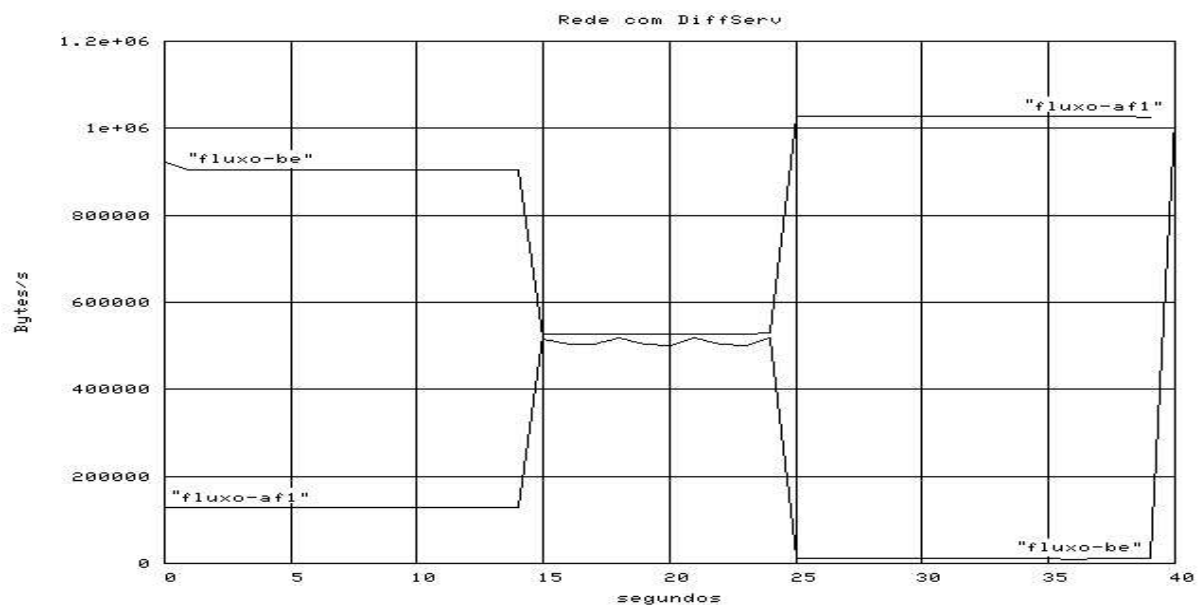


Fig. 5.3: Comportamento da Arquitetura DiffServ para Fluxos das Classes BE e AF.

Inicialmente o “*fluxo-be*” não recebe uma banda maior que 900KBps, uma vez que a banda total é de 1MBps e o “*fluxo-af*” inicialmente submete à rede uma taxa de 100KBps. Como esperado, a medida que a banda de transmissão é requisitada pelo o “*fluxo-af*”, o mesmo valor é decrementado da banda utilizada pelo “*fluxo-be*”. Este exemplo mostra que a configuração feita no domínio DiffServ através do componente Ambiente de Gerenciamento foi respeitada devidamente pelos TCDs.

No exemplo da Fig. 5.4 dois fluxos AF são criados. Inicialmente o fluxo intitulado como “*fluxo-af1*” submete à rede uma vazão de 900KBps, e o fluxo intitulado “*fluxo-af2*” submete à rede uma vazão de 100KBps. No teste, o “*fluxo-af2*” altera sua taxa para 350KBps aos 15 segundos de sua criação e para 500KBps aos 25 segundos.

Como esperado, os fluxos da mesma classe DS são tratados da mesma forma. Este exemplo mostra que a configuração feita no domínio DiffServ através do componente Ambiente de Gerenciamento é respeitada devidamente pelos TCDs.

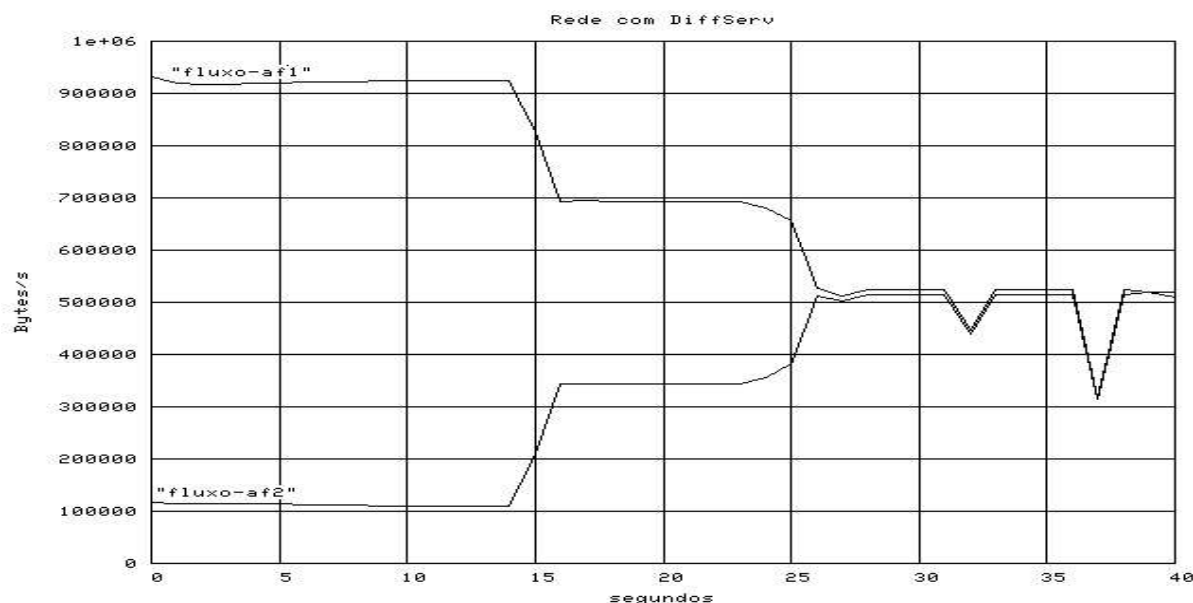


Fig. 5.4: Comportamento da Arquitetura DiffServ para Dois Fluxos da Classe AF.

5.3 Resposta da Arquitetura no Suporte à QoS

Nesta seção o objetivo é apresentar cenários que ilustram as funcionalidades agregadas na Arquitetura DiffServ. A Fig. 5.1 ilustra o ambiente de testes bem como dispõe os componentes TCD, IQoS e o Ambiente de Gerenciamento nos elementos da rede. O componente TCD responsável por configurar o controlador de tráfego está instanciado na máquina *Black* e na *Brutos*. O Ambiente de Gerenciamento, para gerenciar o domínio DS, e o BB, para controlar os recursos, estão instanciados na máquina *Brutos*.

Antes que qualquer classe DS seja criada, é preciso configurar a máquina *Black* para que os valores do campo DSCP (*DiffServ Code Point*) sejam adicionados ao controlador de tráfego desta máquina. O valor do campo DSCP em um pacote indica o tipo do serviço que o pacote receberá, podendo ser descarte, atraso ou precedência em relação aos demais. A Fig. 5.5 mostra o Ambiente de Gerenciamento onde se encontra o botão “Criar Classes TCD de Borda”. Após a seleção deste botão, todos os roteadores de borda são configurados para marcar corretamente os valores do DSCP segundo a especificação DiffServ. Segundo nossa infra-estrutura, apenas a

máquina *Black* é configurada para esta função. A disciplina de fila que possibilita a marcação dos pacotes é a DSMARK.

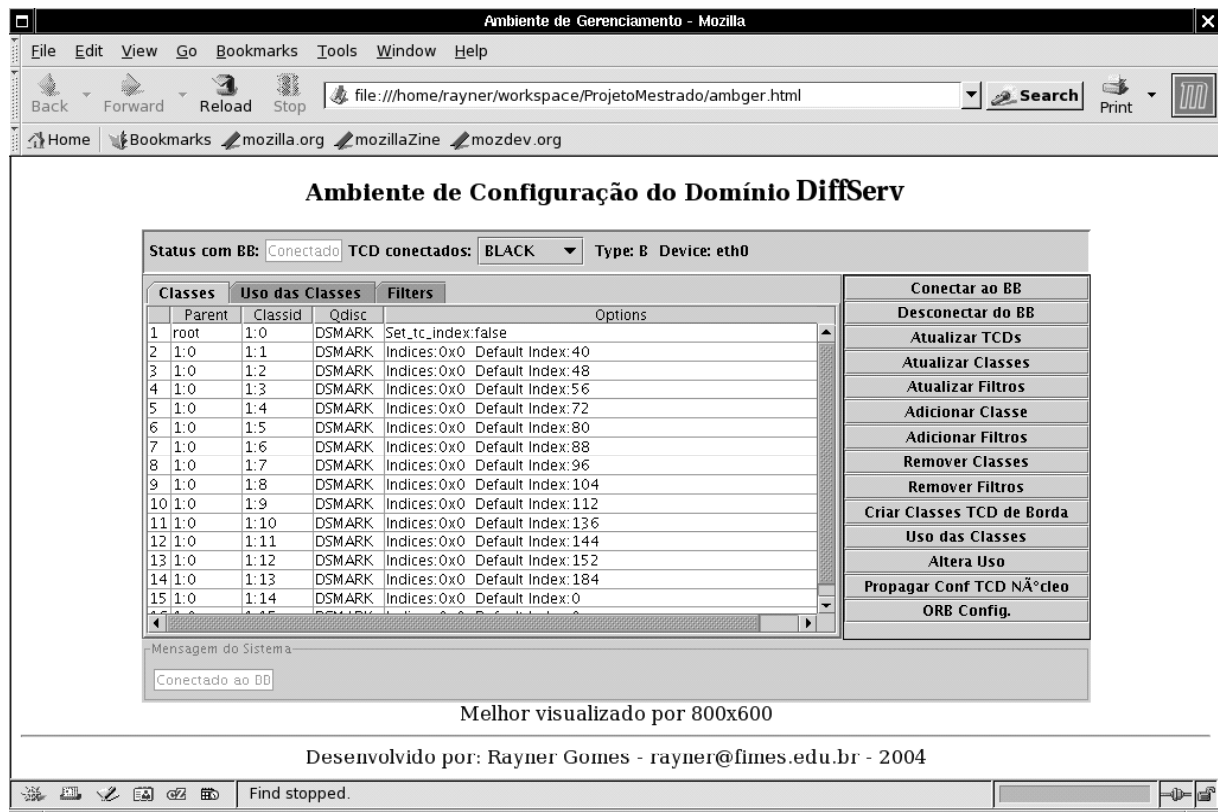


Fig. 5.5: Listagem das Filas DSMARK no Ambiente de Gerenciamento.

Toda a largura da banda de transmissão é limitada a 1Mbps, a classe AF1 possui o valor do DSCP igual a 0x28. Qualquer fluxo com o campo DSCP igual a 0x28 tem toda a banda de transmissão a sua disposição. Fluxos que não forem enquadrados nesta classe terão uma menor prioridade. As classes DSs são implementadas através da QDISC HTB e do filtro TCINDEX do controlador de tráfego presente na máquina Brutos. As QDISCs HTB são criadas pelo administrador do domínio, assim as bandas disponíveis podem ser configuradas de acordo com o interesse do administrador. Uma vez que um pacote com o valor do DSFIELD igual a 0x28 será enquadrado a classDS AF1. Os filtros TCINDEX enquadram um fluxo a uma classe DS no núcleo da rede e são criados automaticamente depois que a classe HTB é criada pelo Ambiente de Gerenciamento.

A Fig. 5.6 apresenta a tela do Ambiente de Gerenciamento que lista todos os TCDs criados nos roteadores. Como já dito nos capítulos anteriores, as classes e filtros do controlador de tráfego são criados nos TCDs selecionados na lista.

Nas próximas seções, apresenta-se cenários com concorrência ou compartilhamento da banda de transmissão utilizando a infra-estrutura mostrada nesta seção e segundo a configurações do domínio DiffServ apresentadas e descritas até aqui.

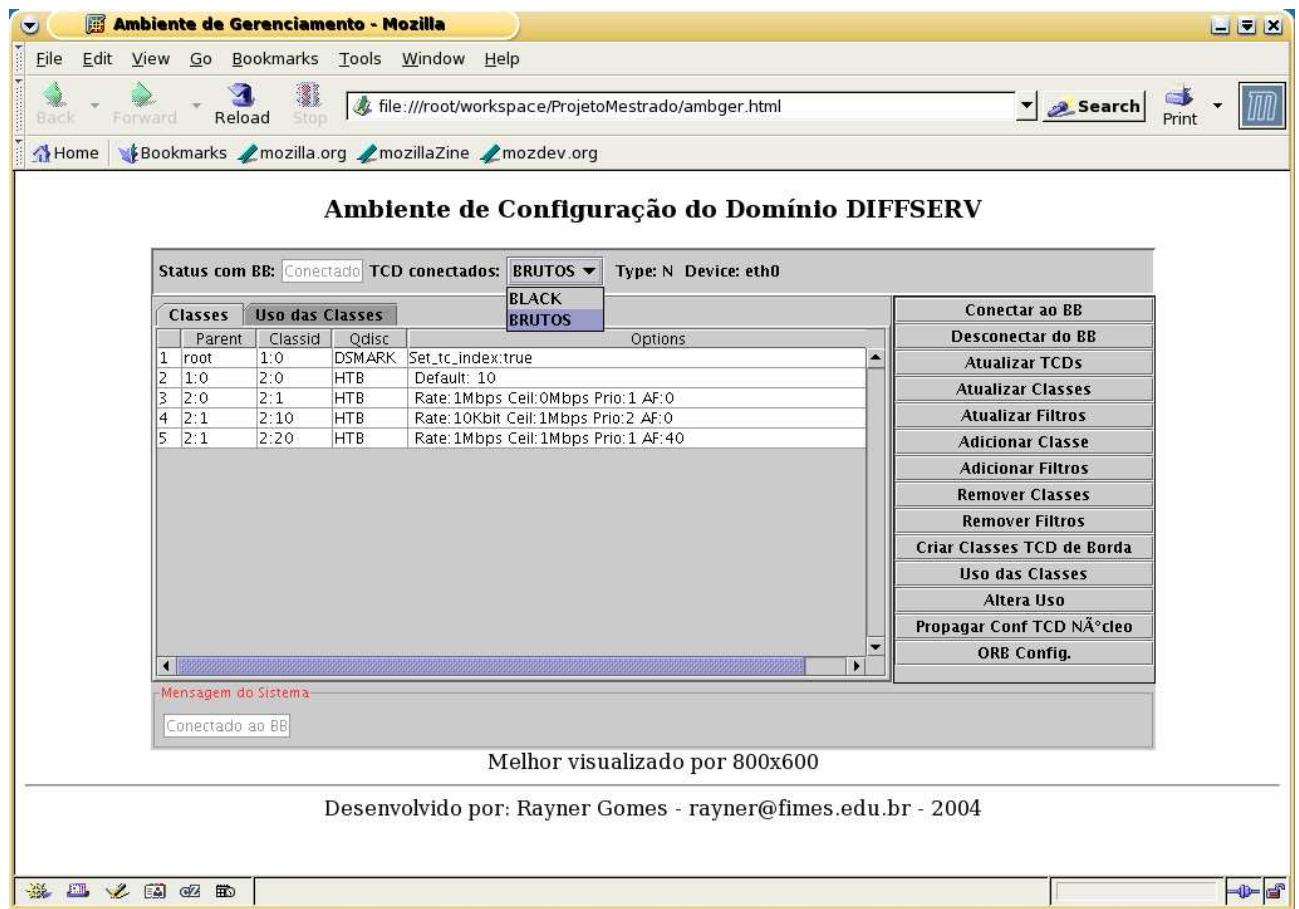


Fig. 5.6: Listagem dos *Daemons* TCDs no Ambiente de Gerenciamento.

5.3.1 Associação de um Fluxo à uma Classe AF

Neste exemplo, dois fluxos associados ao serviço de melhor esforço (BE) e um fluxo associado ao serviço AF são enviados da máquina *Black* para a máquina *Inspiration*. No início do teste os dois fluxos BE utilizam toda a banda de transmissão, o fluxo “*fluxo01-be*” e o “*fluxo02-be*” submetem à rede uma taxa de 700KBps e 300KBps respectivamente. A finalidade do teste é analisar a disponibilização de banda de transmissão independente do quanto os fluxos BE necessitem quando um fluxo associado a classe AF necessita de banda de transmissão.

A seguir, os passos realizados para simular o cenário acima são apresentados:

1. cria-se dois fluxos BE que submetem à rede uma vazão de 700KBps e 300KBps;
2. aplicação requisita uma banda de transmissão de 600KBps;
3. BB verifica quais as classes DS que podem garantir que esta requisição seja atendida;
4. classe DS AF1 é encontrada;

5. BB atualiza a tabela de contabilidade do uso das classes DS;
6. BB requisita a configuração dos roteadores de borda para classificar os fluxos pertencentes a esta aplicação na classe AF1;
7. BB retorna à aplicação que à rede esta configurada para atender a sua requisição de uma banda de transmissão de 600KBps.
8. aplicação que requisitou a banda inicia o fluxo na rede;
9. os dois fluxos BE são penalizados.

O gráfico da Fig. 5.7 mostra que na marca dos 15 segundos a aplicação **IQoSText** cria o “*fluxo03-af*” associado com o serviço AF e logo após a sua criação todos os fluxos BE, *fluxo01-be* e *fluxo02-be* têm a banda degradada pelo “*fluxo03-af*”. O domínio DS neste exemplo se comportou como o esperado.

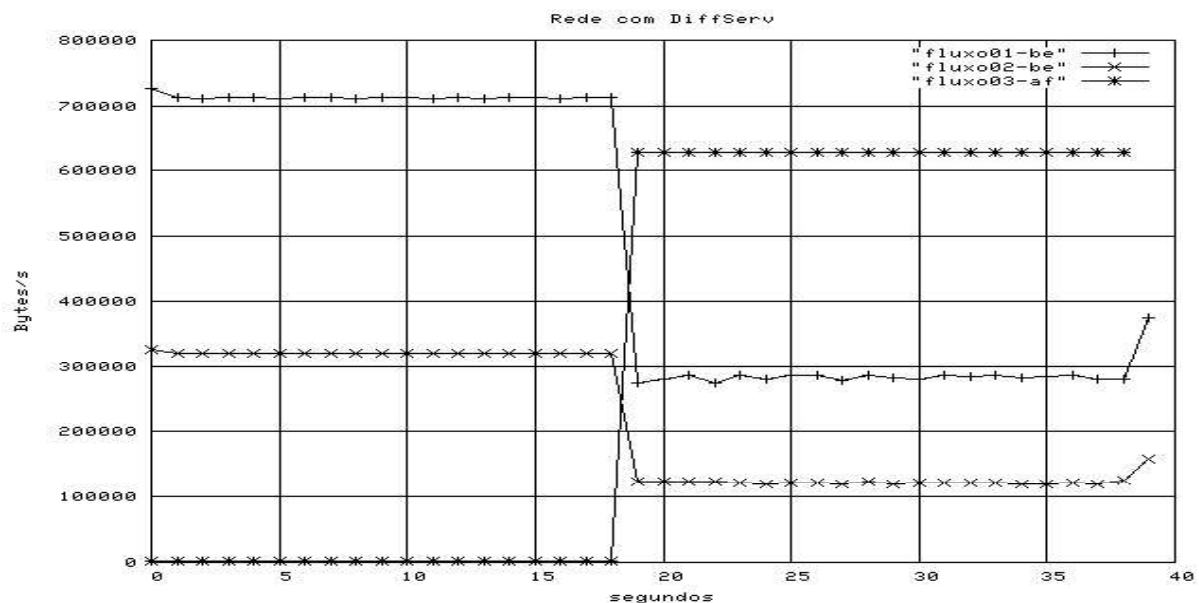
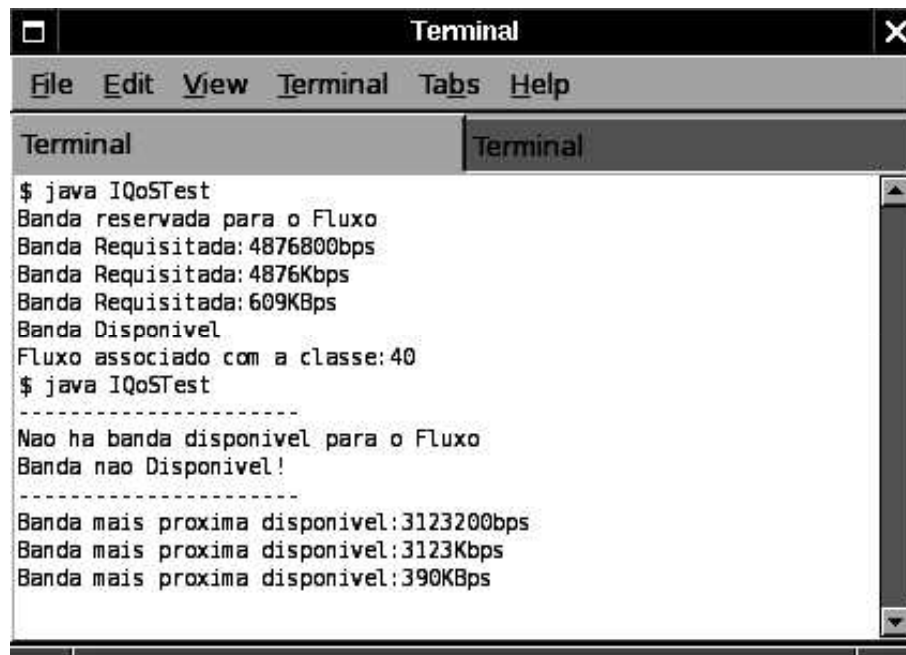


Fig. 5.7: Comportamento de Dois Fluxos BE e um Fluxo AF.

5.3.2 Associações Conflitantes de Fluxos à uma Classe AF

Quando um serviço for garantido a um fluxo, o processo de requisição de banda transmissão por uma aplicação qualquer não poderá afetar o serviço já contratado por uma aplicação. Neste teste são utilizadas duas aplicações na máquina *Black* para simular este cenário, as aplicações **IQoSTest** (Fig. 5.8) são executadas em um terminal sequencialmente. As duas aplicações requisitam juntas uma banda de 609KBps, porém o domínio só pode oferecer 391KBps à segunda aplicação. Neste teste a aplicação que não consegue a banda desejada não requisita renegociação, simplesmente aceita que não há como atender a requisição encaminhada.



```
Terminal
File Edit View Terminal Tabs Help

Terminal
$ java IQoSTest
Banda reservada para o Fluxo
Banda Requisitada: 4876800bps
Banda Requisitada: 4876Kbps
Banda Requisitada: 609KBps
Banda Disponível
Fluxo associado com a classe: 40
$ java IQoSTest
-----
Nao ha banda disponivel para o Fluxo
Banda nao Disponível!
-----
Banda mais proxima disponivel: 3123200bps
Banda mais proxima disponivel: 3123Kbps
Banda mais proxima disponivel: 390KBps
```

Fig. 5.8: Requisição uma Banda não Suportada pela Rede por uma Aplicação.

Os passos que descrevem a requisição de fluxos pelas duas aplicações deste cenário estão detalhados na sequência abaixo.

1. a primeira execução do `IQoSTest` requisita uma banda de transmissão;
2. `IQoS` faz a tradução dos parâmetros de QoS passados pela aplicação e verifica que é necessário uma banda de 609KBps;
3. `IQoS` solicita do BB uma banda de 609KBps;
4. BB verifica em suas classes DS e acha uma classe AF;
5. BB configura todo o domínio DS e retorna uma sinalização positiva para o `IQoS`, a configuração é o processo de sinalizar os TCDs de bordas para realizar o enquadramento do fluxo e atualizar a tabela de contabilidade do uso da classes DS;
6. `IQoS` repassa a sinalização positiva que recebeu do BB para a aplicação que originou a requisição;
7. aplicação instância o fluxo e o mesmo pode fazer o uso da banda assegurada a ele;
8. a segunda execução do `IQoSTest` requisita uma banda de transmissão;
9. `IQoS` faz a tradução dos parâmetros de QoS passados pela aplicação e verifica que é necessário uma banda de 609KBps;

10. IQoS solicita do BB uma banda de 609KBps;
11. BB verifica que não tem banda o suficiente para abrigar ao mesmo tempo as duas aplicações sem que uma não interfira na transmissão da outra e uma sinalização negativa é enviada;
12. BB retorna ao IQoS associado a segunda execução da aplicação a banda disponível;
13. a segunda aplicação é finalizada.

A Fig. 5.9 permite observar que após a marca dos 15 segundos a primeira aplicação inicia o “fluxo03-af” que se mantém constante em 609KBps até o final do teste. Neste período a segunda aplicação não instância nenhum fluxo, posto que a classe AF não suportava outro fluxo de 609KBps. No período em que a segunda aplicação fez a requisição o serviço oferecido à primeira foi honrado.

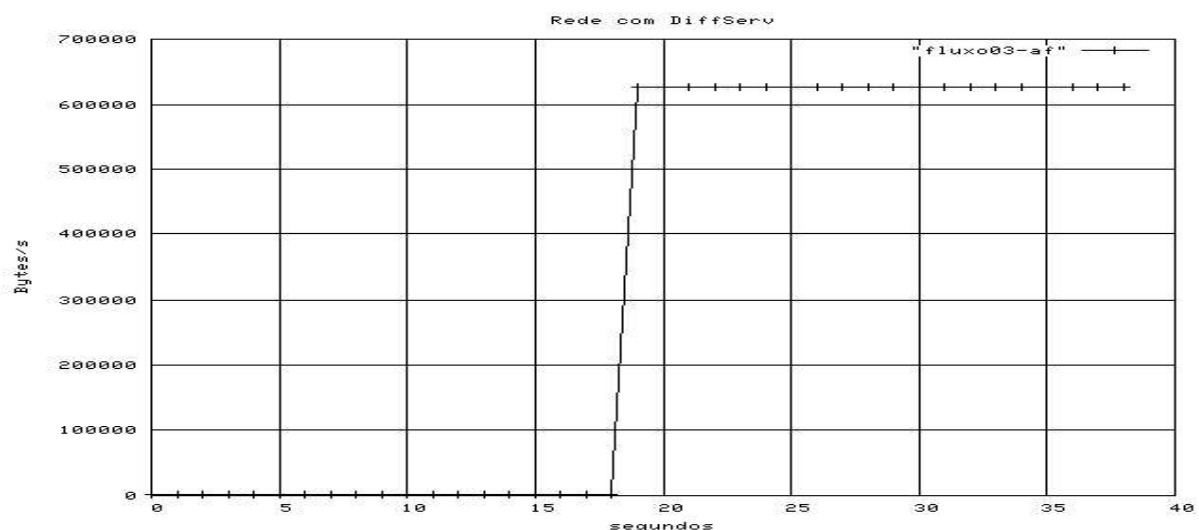


Fig. 5.9: Contabilização da Banda de Transmissão Associada à uma Classe DS

5.3.3 Associação de Dois Fluxos à uma mesma Classe AF

Segundo o cenário anterior, qual seria o comportamento da rede se a segunda aplicação aceitasse a banda oferecida pelo BB? Para simular esta situação o cenário anterior será refeito, porém a segunda aplicação aceita a banda disponível. É esperado que a segunda aplicação utilize apenas os 391KBps oferecidos e que o serviço oferecido a primeira aplicação não seja degradado. Adicionalmente neste teste, é criado um fluxo associado ao serviço BE da máquina *Black à Inspirion* e por todo o período do teste, os quarenta segundos, é esperado que a medida que os dois fluxos associados ao serviço AF sejam instanciados, o fluxo BE seja penalizado, ou seja, a banda fornecida a este fluxo seja retirada e disponibilizada aos outros fluxos.

A seqüência descrita abaixo ilustra o cenário em que a segunda aplicação aceita a banda de transmissão oferecida em detrimento do que foi requisitado, bem como o comportamento dos componentes do Domínio DiffServ.

1. instanciação do fluxo “*fluxo01-be*” submetendo a rede a uma vazão de 1MBps por 60 segundos - este fluxo recebe o serviço de melhor esforço, tendo assim a menor prioridade;
2. fluxo intitulado “*fluxo02-af1*” inicia-se em 9 segundos, este fluxo está associado a primeira execução da aplicação IQoSTest (Fig. 5.8) que requisita da rede uma banda de transmissão de 609KBps;
3. “*fluxo01-be*” é penalizado e recebe não mais do que 400KBps de banda;
4. na marca dos 19 segundos, a primeira execução do IQoSTest inicia-se e o “*fluxo03-af1*” associado a esta aplicação utiliza 390KBps de banda disponibilizada pelo BB;
5. “*fluxo01-be*” é penalizado novamente e recebe não mais do que 100KBps de banda, esse valor é o máximo assegurado aos fluxos associados ao serviço de melhor esforço;
6. depois dos quarenta segundos, a primeira aplicação é finalizada e o “*fluxo02-af1*” é encerrado.
7. a banda liberada é cedida para o “*fluxo01-be*”;
8. finalmente, a segunda execução da aplicação do IQoSTest é finalizada e o “*fluxo03-af1*” é encerrado, observa-se que nos cinqüenta segundos o “*fluxo01-be*” retoma a utilizar toda a banda disponível.

Como pode ser observado na Fig. 5.10, a aplicação instancia nos 10 segundos o fluxo “*fluxo02-af*” que submete a rede a uma taxa de 609KBps. Como decorrência, o fluxo o “*fluxo01-be*” sofre decréscimo de vazão que cai para 400KBps. Aos 19 segundos a segunda aplicação cria o fluxo “*fluxo03-af*”, que submete à rede uma taxa de 300KBps e que faz a vazão do fluxo “*fluxo01-be*” novamente cair para 100KBps. Mesmo após a criação do “*fluxo03-af*” o serviço oferecido ao “*fluxo02-af*” não foi alterado, uma vez que são da mesma classe DS. Aos 40 segundos o “*fluxo02-af*” é encerrado e a banda liberada é cedida para o “*fluxo01-be*”. Aos 50 segundos o “*fluxo03-af*” é finalizado e toda a banda é cedida para o “*fluxo01-be*”. Todo este comportamento vai ao encontro do esperado, mostra-se que toda a configuração do domínio DS foi realizado corretamente por todos os componentes.

5.3.4 Monitoração de uma Classe AF

Os testes anteriores mostraram que os fluxos de mesma prioridade e classe recebem um comportamento de melhor esforço (BE). Uma aplicação que não honrar o contrato e submeter a rede a uma vazão acima da contratada, pode prejudicar o serviço oferecido a outra aplicação.

Para realizar o teste de monitoramento de contratos um segundo ambiente de teste é utilizado. A Fig. 5.11 ilustra este novo ambiente, que diferentemente do primeiro, dispõe de cinco sub-redes que permitem uma melhor distribuição e caracterização do Domínio DiffServ. Uma descrição sucinta das máquinas que o compõe é apresentada a seguir:

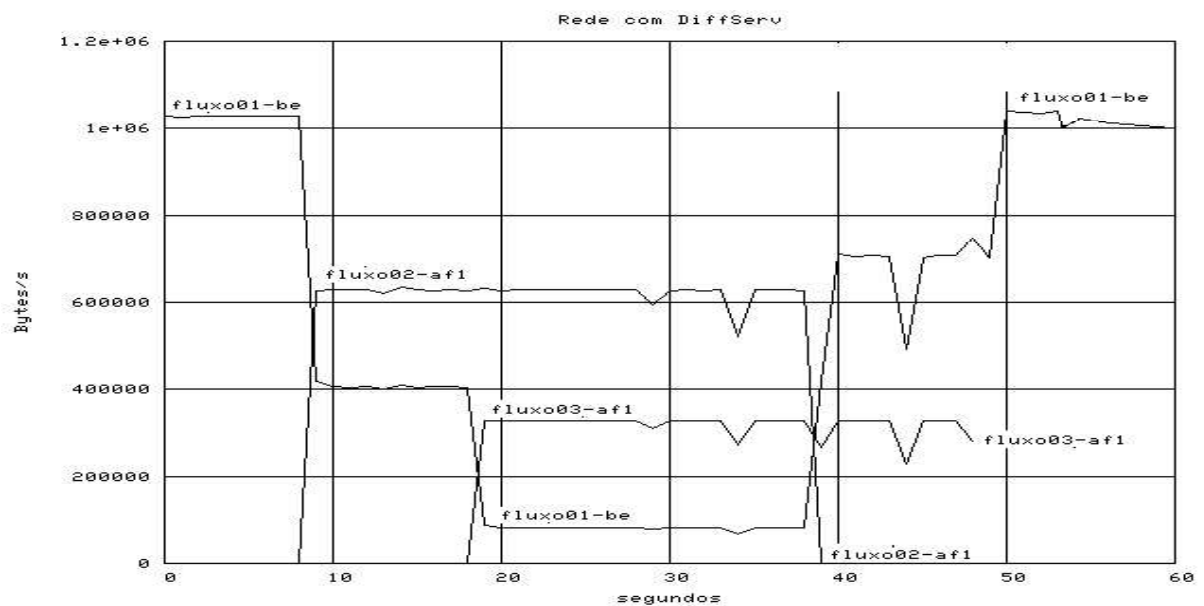


Fig. 5.10: Dinâmica de Dois Fluxos AF e um BE.

- *Cold*: tem a função de receber os fluxos. Os componentes instanciados são: Ambiente de Gerenciamento, BB e o *Monitor-Adapter*;
- *Winter*: tem a função do nó de núcleo. Nela é instanciado o componente TCD;
- *Autumn*: tem a função do nó de borda. Nela é instanciado o componente TCD.
- *Summer*: tem a função de gerar os fluxos. Nela é criada a aplicação IQoSTest e o IQoS;
- *Spring*: tem a função de roteador de rede.

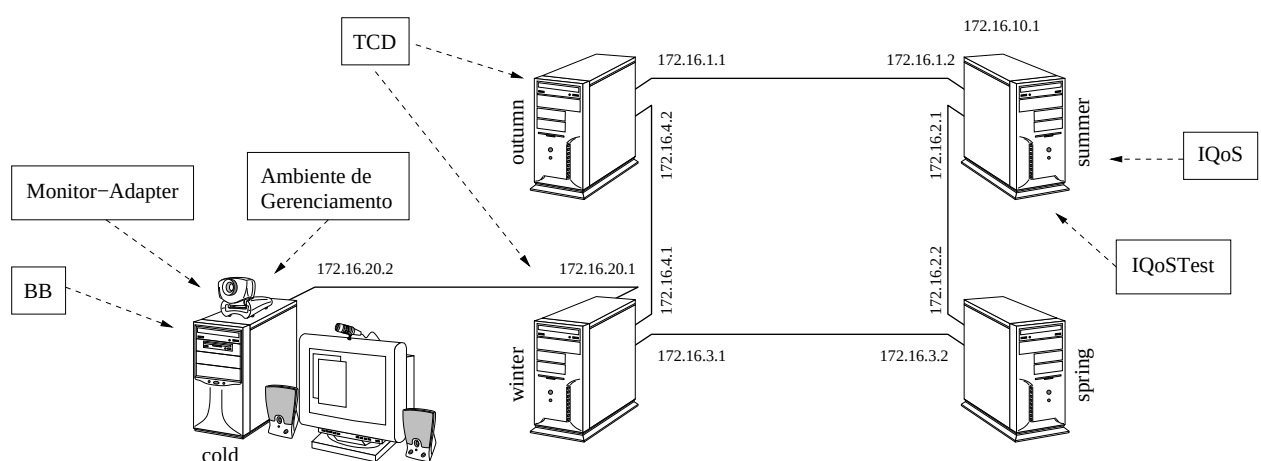


Fig. 5.11: Localização dos Componentes em um Ambiente de Teste com mais Subredes.

Os itens abaixo descrevem as seqüências de execuções das aplicações e o comportamento dos componentes agregados ao domínio DiffServ:

- banda é limitada a 340KBps, sendo que 240KBps é garantida para os fluxos AF;
- durante 60 segundos o “*fluxo03-be*” submete à rede uma taxa de 50KBps;
- de 0 até os 25 segundos o “*fluxo01-af1*” e o “*fluxo02-af1*” submetem à rede uma vazão de 120KBps cada, segundo o contrato estabelecido;
- após os 25 segundos o “*fluxo02-af1*” quebra o contrato e submete à rede uma vazão de 140KBps até o final do teste (60 segundos);
- o “*fluxo01-af1*” é influenciado pela rajada do “*fluxo02-af1*” e acaba sendo penalizado até que o “*Monitor-Adapter*” seja sensibilizado;
- *Monitor-Adapter* detecta a quebra do contrato;
- *Monitor-Adapter* avisa o BB qual a aplicação que quebrou o contrato;
- BB classifica o “*fluxo02-af1*” para receber o serviço BE.

A Fig. 5.12 ilustra o comportamento do “*fluxo02-af1*” que na marca dos 25 segundos quebra o contrato e submete à rede uma taxa maior do que a contratada prejudicando o “*fluxo01-af1*”. Após 5 segundos o *Monitor-Adapter* detecta quebra de contrato e informa o BB qual aplicação é a responsável. Segundo a política de policiamento o BB concede o tratamento BE para o “*fluxo02-af1*”. No momento em que ocorre a mudança de classe o “*fluxo01-af1*” passa a utilizar toda a banda de transmissão da classe.

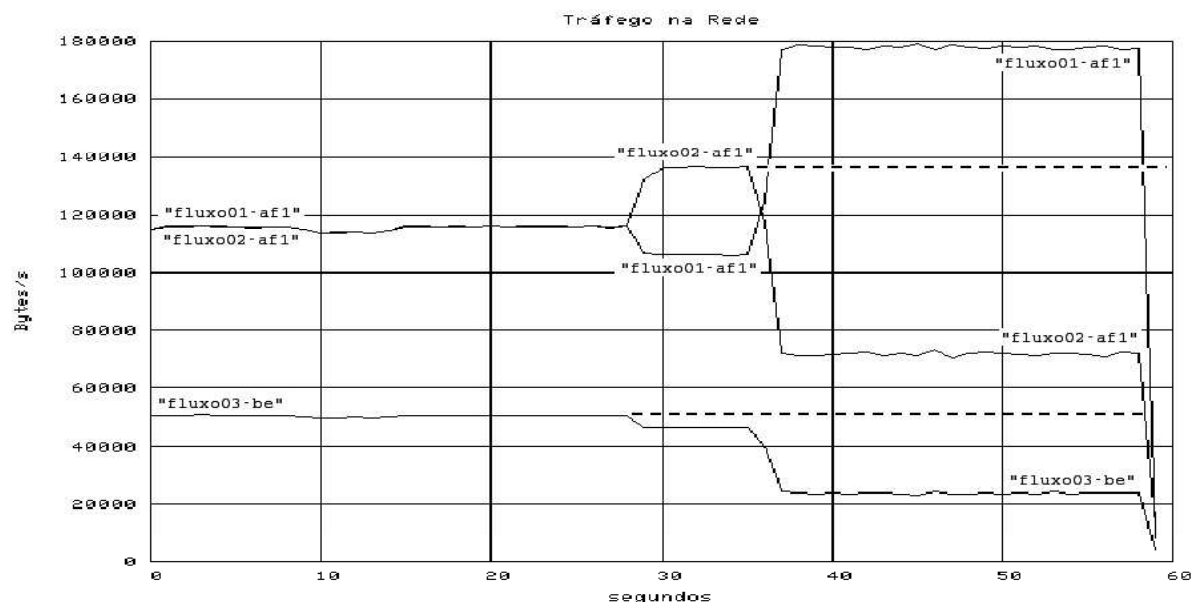


Fig. 5.12: Monitoração de uma Classe AF.

Dois pontos merecem destaque: primeiro, é que apenas o “*fluxo02-af1*” é monitorado pelo *Monitor-Adapter*, assim quando o “*fluxo01-af1*” tem sua vazão alterada ele não é penalizado. O objetivo de não monitorar o “*fluxo01-af1*” é mostrar que após a classificação do “*fluxo02-af1*” da classe AF1 para BE a banda associada a classe AF1 é utilizada pelo “*fluxo01-af1*” sem que este dispute o recurso; segundo, é que uma pequena quantidade de banda é associada ao serviço BE. Durante o teste os componentes precisam se comunicar, a penalização total dos fluxos BE teria como consequência a quebra de comunicação dos componentes envolvidos no teste.

Neste novo ambiente, evidencia-se a importância do componente *Monitor-Adapter* posto que o “*fluxo02-af1*” foi penalizado segundo a política de policiamento previamente estabelecida. A política empregada é um tanto quanto severa, entretanto, outras políticas podem ser configuradas sem alterar a funcionalidade do *Monitor-Adapter*.

Para ilustrar a simulação de fluxos de áudio e vídeo o código abaixo (*IQoSTest*) descreve a instanciação de dois objetos que modelam os parâmetros de QoS associados ao áudio e ao vídeo (linha 01 até 10).

```
01: classAudioFlow audioflow = new classAudioFlow();
02: classVideoFlow videoflow = new classVideoFlow();
03: audioflow.setNumChannel((short) 2);
04: audioflow.setSampleRate((short) 2400);
05: audioflow.setSampleSize((short) 8);
06: audioflow.setQuantization("raw");
07: videoflow.setVideo_colorDepth((short) 16);
08: videoflow.setVideo_sampleRate((short) 20);
09: videoflow.setResolution((short) 200, (short) 100);
10: videoflow.setVideo_colorMode("raw");
```

O código abaixo descreve a instanciação do objeto *IQoS* e a inclusão dos objetos utilizados para modelar uma mídia (linha 01 até 03). Entre as linhas 10 e 12 apresenta-se o código da requisição de banda.

```
01: classIQoS iqos = new classIQoS(args);
02: iqos.avstream.addAudioFlow(audioflow);
03: iqos.avstream.addVideoFlow(videoflow);
04: int resp = iqos.requestBandWidth();
05: if (resp > 0) {
06: if (iqos.getBandwidth() > 0)
07: else {
08: System.out.println("Banda Requisitada:" +
09: iqos.avstream.getSunLenghtBandWidth() + "bps");
10: System.out.println("Banda Disponivel");
11: System.out.println("Fluxo associado com a classe:" + resp);
12: }
```

Na sequência descreve-se como a aplicação *RUDE* através de *scripts* executa o objeto *IQoSText* segundo os parâmetros de QoS da aplicação:

```
01: try { String DIR = "/home/rayner/workspace/ProjetoMestrado";
02: File f = new File(DIR + "/rude-script");
03: FileWriter writer = new FileWriter(f, false);
04: PrintWriter saida = new PrintWriter(writer);
05: saida.println("START NOW");
06: saida.println("0000 0010 ON " +
07: iqos.avstream.getSport() + " " +
08: iqos.avstream.getIpdst() + ":" +
09: iqos.avstream.getDport() + " CONSTANT 100 " +
10: iqos.avstream.getSunLenghtBandWidth() / 800);
11: saida.println(iqos.avstream.getDuration() + "000 0010 OFF");
12: saida.close();
13: writer.close();
14: String COMMAND = "rude -s " + DIR + "/rude-script";
15: System.out.println("Comando:" + COMMAND);
16: if (classUtils.runCommand(COMMAND) < 0)
17:     System.out.println("Erro na chamada do script Rude");
18: } catch (Exception e) {
19:     e.printStackTrace(System.out);
20: }
```

De modo geral, a implementação dos componentes do modelo permitiu constatar a implantação da Arquitetura DiffServ em uma rede de computadores com o *Linux*, e a configuração dos computadores através de objetos distribuídos. Vários testes foram executados através da ferramenta RUDE e ilustrados por meio de gráficos. O BB foi agregado à Arquitetura DiffServ, e segundos os testes, ele permitiu a criação de classes DS e o enquadramento dos fluxos, também o BB fez o controle da quantidade de aplicações associadas a uma classe DS com a finalidade de não sobrecarregá-la.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho realizou-se um estudo sobre a provisão de QoS em aplicações multimídia e como esta provisão pode ser gerenciada através de um ambiente baseado na Arquitetura DiffServ.

Com relação ao escopo e objetivos do trabalho, destacamos alguns pontos que nortearam a proposta apresentada: “*Como uma aplicação pode usufruir da QoS?*”, “*Como negociar uma banda de transmissão?*”, “*Quem deve gerenciar o domínio DS?*”, “*Como implantar o Serviço Diferenciado em uma Rede Local?*”, “*Como enquadrar um fluxo em um Serviço DS?*”

Para tanto, propõe-se e implementa-se um modelo baseado na Arquitetura DiffServ utilizando-se as tecnologias objetos distribuídos e tecnologia Web. O Modelo de Tecnologia é apresentado no Capítulo 4, a Arquitetura de Implementação e Testes no Capítulo 5 possibilitando o rastreamento dos passos e seqüências tomados ao longo de todo este trabalho.

A necessidade de um modelo para prover QoS fica evidente quando se utiliza algum serviço na *Internet* que necessite de garantias para o seu pleno funcionamento. Arquitetura DiffServ é o resultado das experiências obtidas pelo grupo da IETF com a Arquitetura IntServ, que se mostrou não escalonável face à alta sinalização realizada por seus protocolos. A Arquitetura DiffServ apresentou-se mais preparada para ser empregada na *Internet*. Ainda assim, vários trabalhos vêm surgindo o preenchimento de algumas lacunas presentes nessa arquitetura.

Entre os aspectos relevantes desta proposta, destaca-se o dinamismo do componente *Bandwidth Broker* nas configurações dos roteadores. O BB é responsável pelo controle da requisição de QoS analisando o uso de cada classe DiffServ.

Outro aspecto relevante foi a implementação de todos os componentes para testar e avaliar o modelo proposto, procurando comprovar as reais necessidades das inclusões sugeridas. O objetivo foi alcançado quando as necessidades de QoS dos usuários são garantidas sem comprometer nenhum outro usuário. Com o resultado da avaliação do suporte a QoS através do gerenciamento dinâmico das classes DS, as conclusões deste trabalho podem ser descritas assim:

- o *Linux* provê todos elementos necessários para desenvolvimento de projetos neste contexto, em particular o *Kernel* provê recursos para a provisão de QoS através das disciplinas de filas DSMARK, HTB e a GRED e o filtros TCINDEX e U32;
- o controle de admissão é feito por fluxo, mas os recursos são administrados por classe – a falta de recursos não significa a não geração de um fluxo, porém ele recebe um tratamento

BE;

- transparência na comunicação através de uma plataforma de objetos distribuídos em conformidade com a especificação CORBA (Java IDL);
- os componentes do modelo são independentes da plataforma do Sistema Operacional;
- mecanismos para tradução de parâmetros de QoS da camada de aplicação para os de rede e retorno da situação da camada de rede face a requisição;
- administração centralizada dos objetos TCDs distribuídos e classificação dinâmica dos TCDs face às necessidades do administrador através do ambiente de gerenciamento;
- contabilização do uso de classes DS através do BB por meio do ambiente de gerenciamento de modo que a atribuição de um novo fluxo não prejudique nenhum outro associado a mesma classe.

6.1 Trabalhos Futuros

Os componentes desenvolvidos para agregar funcionalidades à Arquitetura DiffServ podem sofrer melhorias e extensões, constituindo em serviços e/ou novas funcionalidades para a Arquitetura DiffServ. Alguns temas para trabalhos futuros são apresentados a seguir:

- **Interação do Monitor-Adapter com o BB** - para que a rede tenha a capacidade de reconfigurar suas classes DS e remover os enquadramentos dos fluxos que a aplicação requisita é necessário que o Monitor-Adapter sinalize o BB;
- **Adição de Conhecimento/Inteligência no BB** - as configurações iniciais das classes DS são criadas conforme o desejo do administrador, pode-se adicionar inteligência no BB para que segundo um conhecimento do uso da rede, toda vez que ele for criado reconfigure automaticamente as classes DS;
- **Configuração de um Perfil de Classes DS** - segundo a especificação da Arquitetura DiffServ, o domínio é um conjunto de máquinas que fornecem o mesmo serviço, este requisito da especificação é alcançado quando todos os roteadores compartilham de uma mesma configuração nos seus controladores de tráfego, mas admitindo que um domínio DiffServ possa ter vários nós de borda, a configuração dos controladores de tráfego tanto na borda como no núcleo devem ser diferentes, e se existirem vários nós de egresso no domínio as classes DS devem ter diferentes bandas de transmissão associadas;
- **Adicionamento de Funcionalidades ao IQoS** - o componente IQoS pode utilizar as propriedades *quantization* do áudio e da codificação do vídeo para calcular a banda total de transmissão;
- **Interceptação de Requisições de QoS das Aplicações Legadas** - criar um *proxy* que possa interceptar a prestação de um serviço de uma aplicação legada e interagir automaticamente com IQoS sem ter a necessidade da aplicação fazer isso.

Bibliografia

- [1] Andrew S. Tanenbaum. *“Redes de Computadores”*. Campus, 2003.
- [2] Robert Haas and Raffael Marty. “Everything over IP, IP over Everything”. White paper, ETH Zurich, Nov 2000. URL at www.docs.uu.se/docs/education/instances/spring2003/datakom_mn/ip-over-anything.pdf.
- [3] Aline C. Viana, Anibal S. Jukemura, Daniela A. Xavier, and Kleber V. Cardoso. “Perspectivas sobre Qualidade de Serviço nos Protocolos da Internet”. *NewsGeneration - Rede Nacional de Ensino e Pesquisa (RNP)*, 4(4), Jul 2000. URL at <http://www.rnp.br/newsgen/0007/art1.html>.
- [4] R. Braden, D. Clack, and S. Shenker. “Integrated Services in the Internet Architecture: an Overview”. Request for Comments RFC 1633, Internet Engineering Task Force, Jun. 1994. URL at <http://www.ietf.org>.
- [5] J. Wroclawski. “The Use of RSVP with IETF Integrated Services”. Request for Comments RFC 2210, Internet Engineering Task Force, Sep. 1997. URL at <http://www.ietf.org>.
- [6] E. Rosen, A. Viswanathan, and R. Callon. “Multiprotocol Label Switching Architecture”. Request for Comments RFC 3031, Internet Engineering Task Force, Jan. 2001. URL at <http://www.ietf.org>.
- [7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. “An architecture for Differentiated Services”. Request for Comments RFC 2475, Internet Engineering Task Force, Dec. 1998. URL at <http://www.ietf.org>.
- [8] Jose Ferreira de Rezende. “Avaliação do Serviço Assegurado para a Diferenciação de Serviços na Internet”. In *XVII Simpósio Brasileiro de Redes de Computadores SBRC’99*, 1999. URL at <http://www.gta.ufrj.br/~rezende>.
- [9] Daniel A. Vivanco and Anura Jayasumana. “Bandwidth Brokering and Dynamic Resource Allocation in DiffServ Domains for Heterogeneous Application”. In *27th Annual IEEE Conference on Local Computer Networks - LCN 2002*, pages 372–384, November 2002.
- [10] Cisco. “Cisco Network Monitoring and Event Correlation Guidelines”. Technical report, Cisco System Inc., 1999. URL at http://www.cisco.com/warp/public/cc/pd/wr2k/tech/cnm_rg.htm.

- [11] Ferguson. P. and G. Huston. “*Quality of Service: Delivering QoS on the Internet and in Corporate Networks*”. Wiley Computer Publishing, 1999.
- [12] B. Teitlebaum and T. Hans. “QoS Requirements for Internet2”. Technical paper, Internet2, 1997. URL at <http://qos.internet2.edu>.
- [13] Stardust. “QoS protocols and architectures”. White paper, Stardust Technologies, Inc., 1999. URL at http://www.qosforum.com/white-papers/qosprot_v3.pdf.
- [14] Stardust. “The Need of QoS”. White paper, Stardust Technologies, Inc., 1999. URL at http://www.qosforum.com/white-papers/Need_for_QoS-v4.pdf.
- [15] Carlos G.A. da Rocha and Guido L.S. Filho. “Framework para provisão de Qualidade de Serviço em Redes IP”. In *II Workshop RNP2*, 2000. URL at <http://www.rnp.br/wrnp2/2000/aceitos.html>.
- [16] D. Grossman. “New Terminology and Clarification for DiffServ ”. Request for Comments RFC 3260, Internet Engineering Task Force, Apr. 2002. URL at <http://www.ietf.org>.
- [17] OMG. “The Audio/Video Stream Specification - Version 1.0”. Document formal/00-01, Object Management Group, Jan. 2000. URL at <http://www.omg.org>.
- [18] Cisco. “Quality of Service Overview”. Technical report, Cisco System Inc., 2004. URL at http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/%qos_c/qcintro.htm.
- [19] Dinesh C. Verma. “*Redes de Distribuição de Conteúdo*”. Campus, 2002.
- [20] K. Nichols, S. Blanke, F. Baker, and D. Black. “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers”. Request for Comments RFC 2474, Internet Engineering Task Force, Dec. 1998. URL at <http://www.ietf.org>.
- [21] V. Jacobson, K. Nichols, and K. Poduri. “An Expedited Forwarding PHB”. Request for Comments RFC 2598, Internet Engineering Task Force, Jun. 1999. URL at <http://www.ietf.org>.
- [22] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. “Assured Forwarding PHB Group”. Request for Comments RFC 2597, Internet Task Force, Jun. 1999. URL at <http://www.ietf.org>.
- [23] F. Baker. “Requirements for IP Version 4 Routers”. Request for Comments RFC 1812, Internet Engineering Task Force, Jun. 1995. URL at <http://www.ietf.org>.
- [24] J. Heinanen and R. Guerin. “A Single Rate Three Color Marker”. Request for Comments RFC 2697, Internet Engineering Task Force, Sep. 1999. URL at <http://www.ietf.org>.
- [25] Sally Floyd and Van Jacobson. “Random Early Detection gateways for Congestion Avoidance ”. *IEEE ACM Transactions on Networking*, 1(4):397–413, 1993.

- [26] Stardust. "Introduction to QoS Policies". White paper, Stardust Technologies, Inc., 1999. URL at http://www.qosforum.com/white-papers/qospol_v11.pdf.
- [27] Xipeng Xiao and Lionel M. Ni. "Internet QoS: A Big Picture". White paper, IEEE Networks, Mar 1999. URL at <http://www.comsoc.org/ni/private/1999/mar/pdf/Xiao.pdf>.
- [28] G. Politis, P. Sampatakos, and I. Venieres. "Design of a Multi-layer Bandwidth Broker Architecture". *Lecture Notes in Computer Science*, vol. 1938, October 2000.
- [29] Rossano P. Pinto, Eliane G. Guimarães, Eleri Cardozo, and Mauricio F. Magalhães. "Incorporação de Qualidade de Serviço em Aplicações Telemáticas". In *XXI Simpósio Brasileiro de Redes de Computadores - SBRC 2003*, pages 331–346, Maio 2003.
- [30] Alexey N. Kuznetsov. "IPROUTET2 Utility Suite Howto". www.linuxgrill.com, Ni₁²-Disponi₂¹el. URL at <http://www.linuxgrill.com/iproute2-toc.html>.
- [31] Ernani Medeiros. "Desenvolvendo Software com UML Definitivo". Pearson Makron Books, 2004.
- [32] Martin Fowler and Kendall Scott. "UML Essencial". Bookman, 2000.
- [33] W. Almesberger. "Linux Network Traffic Control - Implementation Overview". In *5th Annual Linux Expo*, pages 153–164, 1999. URL at <http://www.almesberger.net/cv/papers/tcio8.pdf>.
- [34] OMG. "The Common Object Request Broker: Architecture and Specification - Version 2.3.1". Document formal/99-10-07, Object Management Group, Oct 1999. URL at www.omg.org.
- [35] Sun Microsystems. "Java™ 2 Software Development Kit". Sun Microsystems, 2000. <http://www.java.sun.com/products/>.
- [36] OMG. "Java Language Mapping to OMG IDL". Document formal/99-07-59, Object Management Group, July 1999. URL at <http://www.omg.org>.
- [37] H. M. Deitel and P. J. Deitel. "JAVA Como Programar". Bookman, 2001.
- [38] Cay S. Horstmann and Gary Cornell. "Core Java", volume I - Fundamentos of Core. Makron Books, 2001.
- [39] Cay S. Horstmann and Gary Cornell. "Core Java", volume II - Recursos Avan₂¹dos of Core. Makron Books, 2001.
- [40] OpalSoft. "Practical QoS", 2003. URL at <http://http://opalsoft.net/qos/>.
- [41] W. Almesberger, J. Salim, A. Kuznetsov, and D. Knuth. "Differentiated Services on Linux". Technical Report ietf internet draft, Internet Enginnering Task Force, 1999. URL at <http://www.almesberger.net/cv/papers/18270721.pdf>.

-
- [42] B. Hubert. “*Linux Advanced Routing and Traffic Control HOWTO*”. LARC, 2002. URL at <http://lartc.org/lartc.pdf>.
 - [43] Leonardo Balliache. “*Differentiated Service on Linux HOWTO*”, Aug. 2003. URL at <http://www.opalsoft.net/qos/DS.htm>.
 - [44] M. Devik. “*HTB Linux queuing discipline manual - user guide*”, May. 2002. URL at <http://luxik.cdi.cz/~devic/qos/htb/manual/userrg.htm>.
 - [45] Bruce Perens. “The Open Source Definition”. Technical report, Open Source Initiative, 2004. URL at <http://www.opensource.org/docs/definition.php>.