

Universidade Federal de Uberlândia - UFU
Faculdade de Computação - FACOM

**WebLab SOA no Domínio de Redes de Computadores
para Experimentos DiffServ**

Autor: Lucio Agostinho Rocha

Orientador: Prof. Dr. Luís Fernando Faina

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Mestre em Ciência da Computação. Área de Concentração: **Redes de Computadores**.

Fevereiro de 2009
Uberlândia, MG - Brasil

Dados Internacionais de Catalogação na Publicação (CIP)

R672w	Rocha, Lucio Agostinho, 1982 - WebLab SOA no Domínio de Redes de Computadores para Experimentos DiffServ / Lucio Agostinho Rocha, 2009. 134f. : 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. I. Faina, Luís Fernando. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título. CDU: 681.3.02
-------	---

Elaborado pelo Sistema de Bibliotecas da UFU / Setor de Catalogação e Classificação

Universidade Federal de Uberlândia - UFU

Faculdade de Computação - FACOM

**WebLab SOA no Domínio de Redes de Computadores
para Experimentos DiffServ**

Autor: Lucio Agostinho Rocha

Orientador: Prof. Dr. Luís Fernando Faina

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Mestre em Ciência da Computação. Área de Concentração: **Redes de Computadores**.

Banca Examinadora

Prof. Dr. Luís Fernando Faina – FACOM/UFU (orientador)

Prof. Dr. Eleri Cardoso – FEEC/UNICAMP

Prof. Dr. Paulo Roberto Guardieiro – FEELT/UFU

Profa. Dra. Eliane Gomes Guimarães – CTI Renato Archer

Fevereiro de 2009

Uberlândia, MG - Brasil

Resumo

Agostinho, L.R., “WebLab SOA no Domínio de Redes de Computadores para Experimentos Diff-Serv”. Dissertação de Mestrado - FACOM/UFU, Uberlândia, MG. Fevereiro 2009.

Este trabalho apresenta um laboratório remoto (WebLab) para o ensino prático da disciplina de Redes de Computadores com o uso de experimentos relacionados à tecnologia *DiffServ*. Para isso, é proposta uma implementação da arquitetura *DiffServ* e a sua avaliação no domínio do laboratório. A implementação do WebLab segue a Arquitetura Orientada a Serviços (SOA) e utiliza o paradigma da Computação Orientada a Serviços (SOC) para disponibilizar experimentos através da composição de *Web Services*. Os experimentos contemplam o monitoramento da submissão de fluxos ao domínio e o estabelecimento da gerência intra-domínio com o uso de um *Bandwidth Broker*. O trabalho também apresenta uma infraestrutura de *software* viável e segura para a gerência administrativa e de uso de experimentos em WebLabs.

Palavras-chave: WebLab, DiffServ, Web Service, Bandwidth Broker, SOC, SOA.

Abstract

Agostinho, L.R., “WebLab SOA no Domínio de Redes de Computadores para Experimentos Diff-Serv”. Master Thesis - FACOM/UFU, Uberlândia, MG. Fevereiro 2009.

This work presents a remote laboratory (WebLab) for practical teaching of Computer Networks discipline with the use of experiments related to DiffServ technology. It is proposed a implementation of DiffServ architecture and its evaluation in the laboratory. The WebLab implementation follows the Services Oriented Architecture (SOA) and uses the Service Oriented Computing (SOC) paradigm through experiments to realize Web Services composition. The experiments includes monitoring of flows submission at domain and the stablishing of management intra-domain using a Bandwidth Broker. The work also provides a viable and safe software infrastructure for administrative and use management of experiments in WebLabs.

Keywords: WebLab, DiffServ, Web Service, Bandwidth Broker, SOC, SOA.

“Eu sempre pensei que a computação deveria ser feita para as pessoas, e não o contrário.”

Autoria própria

“Um passo de cada vez.”

Autoria própria

Agradecimentos

À santíssima trindade: Pai, Filho e Espírito Santo, em quem confio e que está sempre presente em cada etapa de minha vida.

Aos meus pais, Naninha e Tatá, pelo amor, união e companheirismo que incentivam meus estudos. Aos meus irmãos, Wagner e Fernanda, que também participam dos momentos mais importantes da minha vida com alegria, fraternidade e perseverança. Aos meus demais familiares que não superam em número o meu sincero agradecimento.

Ao meu orientador, Luís Fernando Faina, pelo seu compromisso com o ensino de qualidade e cumprimento do seu papel de orientador necessários para o sucesso de qualquer trabalho científico. A sua contribuição enquanto pessoa e educador foi refletida no decorrer desse projeto e será lembrada como complemento importante para a minha formação pessoal e profissional.

Aos verdadeiros irmãos que tive a oportunidade de conhecer no mestrado: Italo Tiago, pela sua alegria e companheirismo verdadeiro; Fabíola Bento e Célio Domingues, pela amizade e partilha de conhecimento; agradeço também ao amigo Lucas Scotta por suas valorosas lições de vida; Adriano Fiad, por participar da difícil caminhada e valorizar a contribuição tanto quanto o conhecimento; Ricardo Bortolatto e família, pela hospitalidade e respeito incondicional; também aos amigos João Eurípedes, Valquíria Aparecida, Fernanda Barbosa, Liliane Nascimento e tantos outros.

Ao professor Marcelo Maia por me auxiliar na idéia de que é possível melhorar a qualidade do software com propostas de melhoria da documentação.

Paulo Rodolfo (FEEC/UNICAMP) pela disposição na colaboração e no esclarecimento de dúvidas em momentos importantes desse trabalho.

Aos amigos da secretaria, Maria Madalena, André e Maria Helena, pela fundamental prontidão em atender aos estudantes dessa instituição.

À CAPES pelo apoio financeiro.

Em especial aos amigos do Ministério Universidades Renovadas de Uberlândia (MUR) e ao Grupo de Partilha dos Profissionais (GPP) por mostrarem que é possível mudar o mundo em nós mesmos, com pequenos passos, e fazê-lo um lugar melhor para se viver.

Agradeço também a todos que acreditaram em mim e estiveram presentes no decorrer desse trabalho (eles não sabem o quanto foram importantes), com as várias histórias que se entrelaçaram formando novas redes de amizade porque mais importante do que as folhas das árvores são as suas flores e frutos.

Sumário

Resumo	v
Abstract	vi
Lista de Figuras	xii
Lista de Tabelas	xiv
Glossário	xv
1 Introdução	1
1.1 Motivações	1
1.2 Trabalhos Correlatos	3
1.2.1 WebLabs de Interação com Dispositivos Físicos	3
1.2.2 Implementação da Arquitetura <i>DiffServ</i>	10
1.3 Contribuições	12
1.4 Contexto do Trabalho	13
1.5 Estrutura da Dissertação	13
2 WebLabs de Redes com Suporte a SOA	15
2.1 Requisitos Funcionais para WebLabs de Redes	15
2.2 Visão Geral da Arquitetura SOA	16
2.2.1 <i>Web Services</i>	17
2.2.2 SOAP	18
2.2.3 WSDL	19
2.3 Requisitos de um WebLab de Redes com Suporte SOA	22
2.4 Considerações Finais	23
3 WebLabs de Redes com Suporte a <i>DiffServ</i>	24
3.1 Teoria de Serviços Diferenciados	24
3.2 O Campo DS	26
3.2.1 PHB	27
3.2.2 <i>Assured Forwarding PHB</i>	28
3.2.3 <i>Expedited Forwarding PHB</i>	28
3.2.4 Arquitetura <i>DiffServ</i>	29
3.3 Visão Geral do <i>Bandwidth Broker</i>	30
3.3.1 Gerenciamento de Recursos Intra-domínio	31
3.3.2 Gerenciamento de Recursos Inter-domínios	31
3.4 Controle de Tráfego no Linux	32

3.4.1	<i>Qdisc</i> FIFO	32
3.4.2	<i>Qdisc</i> PRIO	33
3.4.3	<i>Qdisc</i> TBF	34
3.4.4	<i>Qdisc</i> SFQ	35
3.4.5	<i>Qdisc</i> RED	35
3.4.6	<i>Qdisc</i> GRED	36
3.4.7	<i>Qdisc</i> HTB	37
3.4.8	<i>Qdisc</i> DSMARK	38
3.4.9	Policiamento	39
3.5	Requisitos de um WebLab para Experimentos <i>DiffServ</i>	39
3.6	Considerações Finais	40
4	Arquiteturas do NetLab WebLab	41
4.1	Arquitetura SOA do NetLab WebLab	41
4.1.1	Serviços de Acesso	42
4.1.2	Serviços de Interação	42
4.2	Gerência Administrativa	42
4.3	Gerência de Uso dos Experimentos	43
4.4	Comparação das Arquiteturas de Gerência de Serviços	43
4.5	Arquitetura dos Experimentos do NetLab WebLab	44
4.5.1	Composição de serviços em experimentos	46
4.6	Serviços de Interação para Experimentos de Redes	47
4.6.1	Serviço de Recursos	49
4.6.2	Serviço de Enlace	50
4.6.3	Serviço de Sessão de Acesso	51
4.6.4	Serviço de Fábrica de Objetos RMI	52
4.7	Serviços de Interação para Experimentos <i>DiffServ</i>	53
4.7.1	Serviço BB	55
4.7.2	Serviço de Geração de Fluxos	57
4.8	Processos de Início e Término de Experimentos	59
4.8.1	Início de Experimentos	60
4.8.2	Finalização de Experimentos	61
4.9	Considerações Finais	61
5	Implementação e Resultados	65
5.1	Disponibilização de Experimentos no NetLab WebLab	65
5.1.1	Coleta e Representação Virtual de Recursos	66
5.1.2	Infraestrutura do NetLab WebLab	67
5.2	Controle de Tráfego no Domínio <i>DiffServ</i>	68
5.3	Experimento <i>Bandwidth Broker</i>	69
5.3.1	Base de Dados	72
5.3.2	Web Service BB	75
5.3.3	Objeto Servidor <i>ServicoBB</i>	76
5.4	Avaliação do Uso das Heurísticas	78
5.4.1	Teste #1: Vazão dos Fluxos Ultrapassa a Largura de Banda Global	79
5.4.2	Teste #2: Vazão dos Fluxos Compatível com a Largura de Banda Global	82
5.4.3	Teste #3: Admissão de Fluxos Respeitando as Heurísticas	82
5.4.4	Teste #4: Controle da Admissão de Fluxos Respeitando as Heurísticas	85

5.5	Experimento Gerador de Fluxos	85
5.5.1	Teste #1: Análise da Vazão em Fluxos Simulados	90
5.5.2	Teste #2: Análise da Vazão de Fluxos de Vídeo	91
5.6	Negociação de Recursos Intra-domínio	92
5.7	Criação de Novos Experimentos	95
5.8	Considerações Finais	96
6	Conclusão e Trabalhos Futuros	97
6.1	Avaliação das Contribuições	97
6.2	Trabalhos Futuros	99
	Referências Bibliográficas	101
A	Configuração DiffServ com Policiamento do Tráfego de Ingresso	107
B	Implementação dos Experimentos	111
B.1	Exemplo de cliente do Web Service BB	111
B.2	Exemplo de Web Service	112
B.3	Exemplo de Interface de Acesso a Objetos RMI	112
B.4	Fábrica de Objetos RMI - Classe principal	113
B.5	Fábrica de Objetos RMI - método para instanciar objetos	113
C	Monitoramento do Tráfego XML	115

Lista de Figuras

1.1	Custo por Demanda de <i>Bandwidth</i> [4].	2
1.2	Arquitetura iLab do MIT para a Integração de WebLabs [9].	5
1.3	Arquitetura de Interação do WebLab-Deusto [11].	6
1.4	Arquitetura Dupla Cliente-Servidor [13].	7
1.5	Arquitetura Genérica para um WebLab [14].	8
1.6	Modelo de Referência para WebLabs [6].	9
1.7	Arquitetura Multi-Camadas para um Domínio DiffServ [26].	10
1.8	Arquitetura CORBA com Incorporação de QoS [27].	11
1.9	Arquitetura de um BB com Suporte a QoS e SNMP [31].	12
2.1	Visão Geral Simplificada da Arquitetura SOA.	17
2.2	Estrutura Geral da Interface do <i>Web Service</i> BB.	19
3.1	O Formato do Campo DS.	27
3.2	Arquitetura <i>DiffServ</i>	29
3.3	Disciplinas de Fila nos Roteadores de Borda.	33
3.4	Exemplo de Hierarquia na Configuração da <i>Qdisc</i> HTB.	37
4.1	Laboratório e seus Experimentos Cadastrados no Projeto <i>AccessService</i>	43
4.2	Experimentos Disponibilizados para o NetLab WebLab no Projeto <i>NetLabWL</i>	44
4.3	Infraestrutura do WebLab GigaBOT [6].	45
4.4	Arquitetura Parcial dos Projetos <i>AccessService</i> e <i>NetLabWL</i>	46
4.5	Modelo de Blocos para a Realização de Ações em Experimentos.	46
4.6	Arquitetura dos Experimentos no <i>NetLabWL</i>	47
4.7	Composição de Serviços no NetLab WebLab.	47
4.8	Diagrama de Classes para Recursos.	49
4.9	Diagrama de Classes para Enlace.	50
4.10	Diagrama de Classes para Sessão.	52
4.11	Arquitetura para Experimentos <i>DiffServ</i> com Recuperação de Sub-recursos e Enlaces.	54
4.12	Arquitetura para Início e Término de Experimentos no NetLab WebLab.	60
4.13	Diagrama de Classes para a Fábrica de Objetos RMI.	62
4.14	Diagrama de Classes para o Experimento Gerador de Fluxos.	63
4.15	Diagrama de Classes para o Experimento BB.	64
5.1	Tags para Indicar os Sub-recursos (NICs) de Cada Recurso (<i>host</i>).	66
5.2	Tags para Indicar os Enlaces entre as NICs dos <i>Hosts</i>	67
5.3	Rede Física do Laboratório.	68
5.4	Rede Lógica do Laboratório.	69
5.5	Arquitetura do Experimento <i>Bandwidth Broker</i>	70

5.6	Experimento BB no NetLab WebLab.	71
5.7	Vazão e Perdas Obtidos com a Garantia de Vazão para Cada Agregado.	81
5.8	Vazão e Perdas Obtidos com o Aumento da Largura de Banda.	83
5.9	Vazão e Perdas Obtidos com a Atribuição de Heurísticas.	84
5.10	Vazão e Perdas Obtidos com Dois Agregados EF Concorrentes.	86
5.11	Arquitetura do Experimento Gerador de Fluxos.	87
5.12	Diagrama de Seqüência do Experimento Gerador de Fluxos.	88
5.13	Experimento Gerador de Fluxos.	89
5.14	Vazão e Perdas para a Submissão do Tipo AF11.	90
5.15	Vazão e Perda sem Diferenciação de Serviços.	90
5.16	Vazão e Perda com Diferenciação de Serviços.	91
5.17	Vazão do Fluxo de Vídeo com Ausência e Presença de Monitoramento.	92
C.1	Monitoramento de Mensagens SOAP.	116
C.2	Monitoramento de Mensagens XML REST.	117
C.3	Monitoramento de Mensagens XML REST Compactadas.	117

Lista de Tabelas

3.1	Prioridade de descarte de pacotes das classes AF.	28
3.2	Unidades utilizadas pelo comando <i>tc</i>	32
3.3	Interpretação das operações lógicas no campo ToS com o comando <i>tc</i>	34
3.4	Distribuição esperada de recursos para os pacotes das classes AF.	36
3.5	Resultado da aplicação das fórmulas para a <i>qdisc</i> GRED.	36
4.1	Relacionamento entre Web Services e Experimentos.	48
5.1	Características dos computadores do NetLab WebLab.	67
5.2	Tabela PHB.	72
5.3	Tabela PHB_EXPERIMENTO.	73
5.4	Tabela SLA.	73
5.5	Tabela RAR.	74
5.6	Tabela EXPERIMENTO_TC.	74
5.7	Tabela CLASS_HTB.	75
5.8	Tabela BB.	75
5.9	Exemplo de distribuição de banda entre os PHBs.	77
5.10	Distribuição de Largura de Banda para cada PHB.	79
5.11	Vazão Gerada por Cada Agregado ao Longo do Tempo.	80
5.12	Redistribuição de largura de banda para os agregados.	82
5.13	Fluxos gerados pela aplicação Gerador de Fluxos.	90
5.14	SLA dos experimentos A e B.	93
5.15	Exemplo de negociação de recursos intra-domínio.	94

Glossário

AF *Assure Forwarding*

AJAX *Asynchronous Javascript with XML*

AMW *Application Middleware*

BA *Behavior Aggregate*

BAR *Bandwidth Allocation Request*

BB *Bandwidth Broker*

BE *Best-Effort*

BP4WS *Business Process Execution Language for Web Services*

CORBA *Common Object Requesting Broker Architecture*

CTI Renato Archer *Centro de Tecnologia da Informação Renato Archer*

DCOM *Distributed Component Object Model*

DiffServ *Differentiated Services*

DP *Drop Precedence*

DS *Differentiated Services*

DS field *Differentiated Services field*

DSCP *Differentiated Services Codepoint*

DTC *Daemon Traffic Control*

ECN *Explicit Congestion Notification*

EF *Expedited Forwarding*

FIFO *First-in, First-out*

FTP *File Transfer Protocol*

GRED *Generalized Random Early Detection*

HTB *Hierarchical Token Bucket*

HTTP *Hypertext Transfer Protocol*

HTTPS *HyperText Transfer Protocol Secure*

IETF *Internet Engineering Task Force*

IG *Interface de Gerência*

IntServ *Integrated Services*

IP *Internet Protocol*

IQoS *Interceptor of QoS*

ISP *Internet Service Provider*

JNLP *Java Network Launching Protocol*

JRE *Java Runtime Environment*

JSP *Java Server Pages*

JWS *Java Web Start*

LDAP *Lightweight Directory Access Protocol*

MIB *Management Information Base*

MIME *Multipurpose Internet Mail Extensions*

MIT *Massachusetts Institute of Technology*

MPLS *Multi Protocol Label Switching*

NIC *Network Interface Card*

PHB *Per-Hop Behavior*

PLD *Programmable Logical Device*

PNG *Portable Network Graphics*

QDisc *Queue Discipline*

QName *Qualified Name*

QoS *Quality of Service*

RAR *Resource Allocation Request*

RCA *Resource Control Agent*

RCL *Resource Control Layer*

RCP *Resource Control Point*

RED *Random Early Detection*

REST	<i>Representational State Transfer</i>
RMI	<i>Remote Method Invocation</i>
RPC	<i>Remote Procedure Call</i>
RSVP	<i>Resource Reservation Protocol</i>
SFQ	<i>Stochastic Fairness Queuing</i>
SLA	<i>Service Level Agreement</i>
SLS	<i>Service Level Specification</i>
SMI	<i>Services Management Interface</i>
SMTP	<i>Simple Mail Transfer Protocol</i>
SNMP	<i>Simple Network Management Protocol</i>
SOA	<i>Service-Oriented Architecture</i>
SOAP	<i>Simple Object Access Protocol</i>
SOC	<i>Service Oriented Computing</i>
SSH	<i>Secure Shell</i>
TBF	<i>Token Bucket Filter</i>
TC	<i>Traffic Control</i>
TCA	<i>Traffic Conditioning Agreement</i>
TCP	<i>Transmission Control Protocol</i>
ToS	<i>Type of Service</i>
UDDI	<i>Universal Description, Discovery, and Integration</i>
UDP	<i>User Datagram Protocol</i>
UML	<i>Unified Modeling Language</i>
URI	<i>Uniform Resource Identifier</i>
URL	<i>Uniform Resource Locator</i>
URN	<i>Uniform Resource Name</i>
VoIP	<i>Voice over IP</i>
VQ	<i>Virtual Queue</i>
WSDL	<i>Web Service Description Language</i>
WWW	<i>World Wide Web</i>
XML	<i>Extensible Markup Language</i>

Capítulo 1

Introdução

Este capítulo apresenta as motivações que levaram à confecção de um WebLab de ensino de Redes de Computadores com foco na área de Serviços Diferenciados. A seguir são apresentados os trabalhos correlatos e as contribuições alcançadas com destaque para os objetivos propostos. Finalmente são descritos o contexto do trabalho e a estrutura da dissertação.

1.1 Motivações

Uma rede de computadores permite que diversos usuários utilizem aplicações para a troca de mensagens, acesso remoto a outros computadores e demais nós da rede, para *upload* e *download* de arquivos, entre muitos outros serviços. A Internet é uma rede que possui diversos usuários utilizando diferentes aplicações ao redor do globo.

Cada um desses usuários utiliza aplicações que exigem diferentes recursos da rede, mas a maioria dessas aplicações estabelece a comunicação informando o endereço IP (*Internet Protocol*) e a porta remota sem levar em consideração outras necessidades do aplicativo e do usuário. Essa comunicação é realizada segundo os protocolos da arquitetura Internet e, por muito tempo, a garantia de acesso com a velocidade contratada era o quesito suficiente para atender às necessidades da maioria desses usuários. Os esforços em pesquisa e desenvolvimento da arquitetura Internet se preocuparam em tornar a plataforma robusta e flexível para as aplicações [1].

A Qualidade de Serviço (*Quality of Service - QoS*) da rede é um fator importante para atender às necessidades das aplicações. QoS pode ser definida como um conjunto de requisitos garantidos para as aplicações quanto ao transporte de fluxos na rede [2]. Dentre os requisitos mais comumente aceitos para descrever a QoS podem ser citados o atraso, perda de pacotes, *jitter* e taxa de erros.

Atualmente, a Internet não faz distinção entre os tipos de tráfego e seus requisitos. O serviço de encaminhamento de pacotes oferecido é do tipo melhor-esforço, ou seja, não há garantias quanto à entrega de pacotes fim-a-fim. Os pacotes são encaminhados pelos roteadores de acordo com o algoritmo FIFO (*First-in, First-out*) com o simples descarte dos pacotes que excederem o *buffer* das interfaces de rede nos nós intermediários. Para as aplicações de tempo-real, multimídia, ensino à distância, acesso e controle de instrumentos remotos, visualização científica, entre outros, são exigidos mais recursos da rede para garantir a qualidade fim-a-fim da comunicação [3]. Nesse contexto, a diferenciação dos serviços otimiza o uso da rede para atender às necessidades de QoS das aplicações.

Um quesito importante para administradores de rede e que também está relacionado à QoS diz respeito ao compartilhamento de largura de banda (*bandwidth*) para diferentes classes de tráfego. Cada uma dessas classes pode oferecer diferentes garantias para o encaminhamento dos pacotes que trafegarem por elas. Essa solução é uma alternativa para prover o uso mais eficiente da rede, ainda que a capacidade do enlace seja fundamental para oferecer serviços de melhor qualidade.

A abordagem de Serviços Diferenciados (*Differentiated Services - DiffServ*) ganhou destaque nos últimos anos por permitir o controle de fluxos de rede com uma escalabilidade maior se comparada às demais propostas [3]. No entanto, a avaliação das configurações fica a cargo do administrador de redes que escolhe as ferramentas que julgar mais adequadas para avaliar os resultados.

A diferenciação de serviços reduz o desperdício de uso da largura de banda e é uma alternativa para otimizar a distribuição de recursos na rede [4]. A Fig. 1.1 ilustra um cenário onde o ISP (*Internet Service Provider*) atribui um custo fixo c para o uso x_u da banda. p representa o valor cobrado proporcional à unidade de uso, medida em Megabytes (MB) de dados transferidos ou minutos de tempo de conexão. A demanda do usuário é modelada como uma função $D(p) = x_u$ e o custo absoluto relativo ao uso de recursos pode ser calculado como a área cx_f , onde x_f é a situação onde o usuário não consome recursos da rede ($D(0) = x_f$). No entanto, a demanda do usuário não é constante: para as oscilações que ocorrem no intervalo entre x_u e x_f o custo efetivo é o resultado da soma de todos os custos das n oscilações em um determinado período de conexão, ou seja, $\sum_{i=1}^n c(x_f^i - x_u^i)$. Esse custo é inferior ao custo fixo cobrado pelo ISP. A conclusão é que mesmo não utilizando os recursos da banda, o custo associado à demanda permanece constante, com um desperdício da alocação de recursos de rede para o usuário.

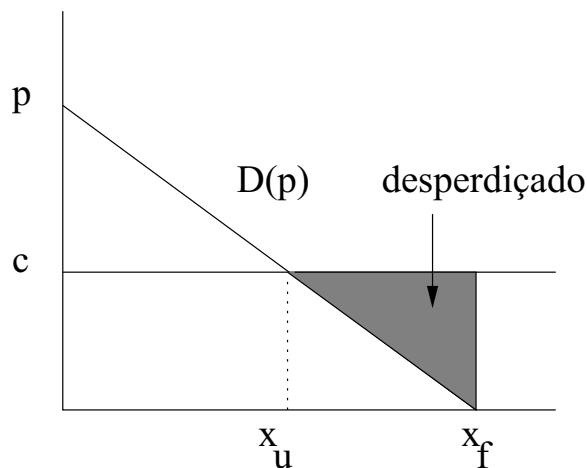


Fig. 1.1: Custo por Demanda de *Bandwidth* [4].

O ensino de *DiffServ* geralmente envolve a explicação de sua arquitetura. O estudo mais rigoroso exige o conhecimento dos comandos e dos algoritmos das ferramentas que implementam o controle de tráfego. Para verificar a dinamicidade da configuração e avaliar os resultados da diferenciação de serviços é necessário um ambiente com dispositivos gerenciáveis que suportem o uso da tecnologia.

Por isso, a utilização de um laboratório de redes de computadores que permita a realização de diversos experimentos de rede, entre os quais, experimentos relacionados à tecnologia *DiffServ* é um grande atrativo. Diante disso, é proposta a configuração física e lógica de um domínio *DiffServ* que permita analisar o comportamento do fluxo de pacotes intra-domínio. A tecnologia *DiffServ* foi a alternativa adotada para o provimento de *QoS* para os experimentos de rede do laboratório. Ela possibilita a agregação de fluxos de necessidades semelhantes em classes de comportamento. Cada classe define parâmetros de *QoS* para tratar o encaminhamento de pacotes de acordo com diferentes níveis de prioridade. Para que os nodos do laboratório reflitam as mesmas configurações diante da submissão de tráfego, foi desenvolvido um *Bandwidth Broker* [5].

Para atender um grande número de estudantes e otimizar a utilização dos recursos do laboratório foi proposta a integração com a Web na forma de um WebLab que recebeu o nome de NetLab

WebLab. Esse WebLab oferece um conjunto de experimentos de interação com dispositivos remotos gerenciáveis. Dentre os experimentos de rede são destacados os experimentos de QoS utilizando a abordagem *DiffServ*.

A infraestrutura de *software* para administração e gerência de acesso aos experimentos utiliza parte das aplicações Web do projeto GigaBOT [6] proposto pela Faculdade de Engenharia Elétrica e de Computação (FEEC) da Universidade Estadual de Campinas (UNICAMP). Essas aplicações foram modificadas neste trabalho para adaptá-las quanto aos quesitos de portabilidade, manutenção e escalabilidade no domínio de redes de computadores. Do conjunto de aplicações Web foram utilizados os aplicativos de gerência administrativa e de uso dos experimentos.

O aplicativo de gerência administrativa reúne informações para a gerência de WebLabs, disponibilização de experimentos e recursos, cadastro de usuários, experimentos e recursos, disponibilização de experimentos em datas e horários previamente agendados, entre outras funções administrativas.

O aplicativo de gerência de uso exibe e permite o acesso aos experimentos reservados para o usuário autenticado. Uma vez que o experimento tenha sido disponibilizado no aplicativo Web de gerência administrativa, o usuário pode realizar a reserva de uso do experimento no horário e data que considerar mais conveniente.

Nesse trabalho, buscou-se implementar uma arquitetura *DiffServ* que permitisse adquirir experiência com a abordagem com a análise dos resultados de submissão de fluxos no ambiente do WebLab. A seguir buscou-se integrar o laboratório com a Web com o uso de experimentos de rede voltados para *DiffServ*. Finalmente, foram investigadas soluções para viabilizar o desenvolvimento e o uso dos experimentos remotamente.

1.2 Trabalhos Correlatos

O contexto da dissertação faz a integração de experimentos relacionados a *DiffServ* em um WebLab de experimentação remota. Por isso, os trabalhos correlatos foram divididos em duas sessões: WebLabs de interação com dispositivos físicos e implementação da arquitetura *DiffServ*. A seleção dos trabalhos sobre WebLabs foi baseada na portabilidade das arquiteturas descritas, na flexibilidade para adição de novos experimentos e *hosts* gerenciáveis, no uso de tecnologias de código-fonte aberto tanto para a confecção dos WebLabs quanto para a interação com os dispositivos remotos. No contexto *DiffServ* foram escolhidos trabalhos relevantes que apresentam modelos para a gerência e interação remota com os *hosts* do domínio com o uso do elemento *Bandwidth Broker*.

1.2.1 WebLabs de Interação com Dispositivos Físicos

Um WebLab é um laboratório remoto controlado através da Internet [7]. O laboratório geralmente é formado por uma infraestrutura de *hardware* e *software* que permite a interação do usuário com os recursos remotos. Esses laboratórios podem ser utilizados para realizar experimentos que possuem maiores exigências quanto à configuração do ambiente de testes. Essa característica também exige um estudo mais cuidadoso da arquitetura de comunicação utilizada de forma que a experimentação remota não seja prejudicada em função da qualidade do enlace ou da tecnologia de interação entre o usuário e o laboratório.

Em eletrônica e automação a seguinte classificação é sugerida para WebLabs [8]:

- WebLabs de instrumentação remota: experimentos onde o usuário atua como um observador capaz de interagir com entradas pré-definidas e visualizar as saídas reais ou virtuais. É possível modificar os parâmetros que afetam o experimento e avaliar os resultados dessa modificação, ou apenas visualizar as medidas resultantes da experimentação;

- WebLabs de controle remoto de parâmetros: o usuário é capaz de alterar as entradas e os parâmetros que afetam a lógica do sistema. Usualmente o usuário é capaz de gravar alguns dados remotamente e ler as respostas do experimento;
- WebLabs de controle lógico remoto: o usuário tem maior nível de controle sobre os recursos do WebLab, ou seja, o usuário pode alterar as entradas, modificar os parâmetros do sistema e alterar a lógica da experimentação.

Os WebLabs de experimentação remota permitem que estudantes acessem via TCP/IP (*Transmission Control Protocol/Internet Protocol*) o equipamento de *hardware* e programas, e desta forma eles monitoram e controlam a real evolução de seu caso prático com uma *webcam*, ou por quaisquer outros meios [7]. Esses laboratórios diferem dos WebLabs de simulação e de experimentação virtual que utilizam exclusivamente *softwares* para representar os recursos e os resultados dos testes. As características inerentes dos laboratórios de experimentação remota exigem que o acesso ao laboratório seja controlado e que o *hardware* do ambiente de testes seja pré-configurado antes de iniciar qualquer experimento. O controle de acesso é feito para garantir que apenas um usuário tenha acesso aos recursos físicos por vez para evitar inconsistência nos resultados. Já a pré-configuração garante um início consistente dos recursos físicos que serão disponibilizados. Ainda que apenas um usuário tenha acesso por vez a um experimento outros estudantes podem acompanhar o curso dessa atividade com uma *webcam* ou qualquer outro serviço de difusão de informações que não interfira diretamente na experimentação atual.

Sempre foi um objetivo das universidades descentralizar parte de suas atividades e encorajar grupos de trabalho ou trabalho colaborativo: trazendo a universidade para todos e em qualquer lugar [7]. Em algumas áreas de ensino como as redes de computadores, por exemplo, a preparação do ambiente de estudo prático para lidar com recursos físicos exige esforço para a elaboração de cenários relevantes. Nesse caso, a experimentação remota apresenta a vantagem de promover o estudo colaborativo com um conjunto de experimentos pré-configurados.

Por outro lado, a simples interação do usuário com o laboratório remoto não constitui uma forma de aprendizagem. Os experimentos devem ser elaborados de maneira que a atividade possa ser didática, mas que destaque também as questões que surgem na interação com o *hardware* do experimento remoto e que são limitadas pelo ensino não-presencial como, por exemplo, o tempo de submissão do resultados, a *performance* dos recursos, o tempo de resposta dos testes, as características do enlace da rede interna do laboratório, entre outros. A própria elaboração da arquitetura do *software* de interação com o laboratório intra e inter-domínio precisa considerar a qualidade do serviço oferecido fim-a-fim.

Os WebLabs de experimentação remota oferecem uma alternativa viável para o ensino à distância, a experimentação e o acesso a material de pesquisa selecionado. Apesar da riqueza de se prover um laboratório com acesso aos recursos físicos à distância, surgem também outras alternativas como os laboratórios virtuais e laboratórios de simulação [7].

Os requisitos funcionais dos laboratórios virtuais e de simulação são diferentes dos laboratórios de experimentação remota “real” e, por isso, a comparação entre as soluções das abordagens não é bem definida. Ainda assim, os laboratórios de experimentação com dispositivos físicos apresentam a vantagem de exibir um resultado mais preciso e de permitir o acesso seguro de um grande número de usuários a recursos distantes que, em alguns casos, são de elevado valor financeiro. Essas características são um grande atrativo.

Dentre os vários projetos de WebLabs, o projeto iLab [9] do MIT (*Massachusetts Institute of Technology*), foi selecionado nesse contexto por se destacar na proposição de laboratórios reais acessíveis através da Internet. O projeto iLab pretende disseminar tecnologias e pedagogias com diversos laboratórios *online* (“iLabs”). O projeto levou dois anos para o desenvolvimento, teste e implementação

dos WebLabs. Atualmente busca-se manter a infraestrutura de *software* obtida para que os laboratórios estejam disponíveis 24 horas/dia x 7 dias/semana e que possam compartilhar recursos com outros laboratórios além das fronteiras do MIT.

Para isso, o grupo de pesquisa desenvolveu um *toolkit* de módulos reutilizáveis, um conjunto de protocolos padronizados e *Web Services*. Todo esse conjunto formou um *framework* de *software* conhecido *iLab Shared Architecture*. Esse *framework* separa os laboratórios em três módulos conectados por uma arquitetura de *Web Services*, como ilustrado na Fig. 1.2. Os módulos dessa arquitetura possuem as seguintes características:

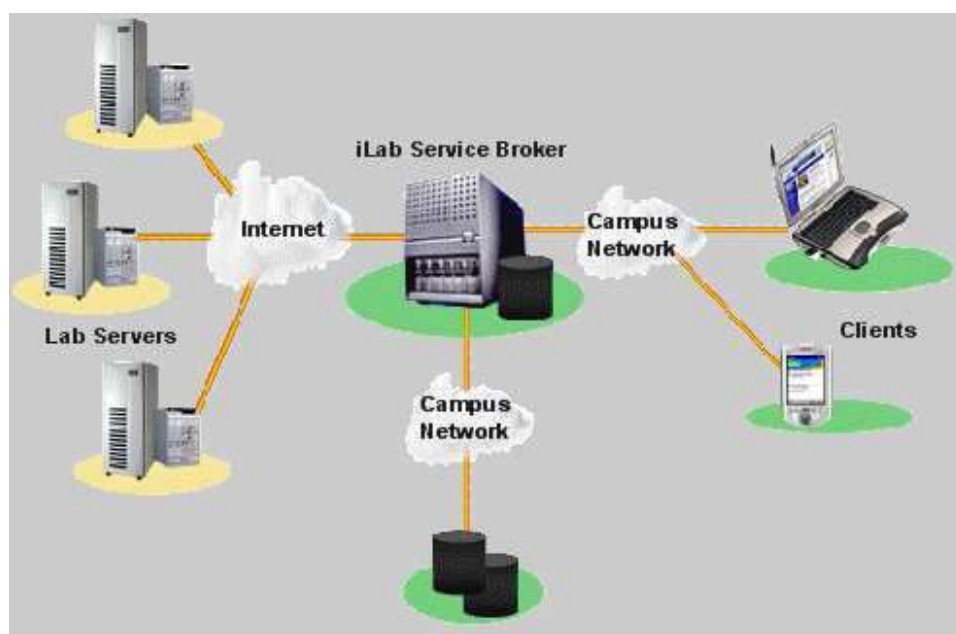


Fig. 1.2: Arquitetura iLab do MIT para a Integração de WebLabs [9].

- *Lab Server* - controlado pelo administrador do laboratório e contém a infraestrutura para controlar e receber dados de/para a experimentação.
- *Lab Client* - executa na máquina do usuário e fornece a interface de interação com o laboratório.
- *Service Broker* - faz o intermédio de comunicação entre os módulos *Lab Client* e *Lab Server* e oferece serviços de persistência e administrativos.

O projeto iLab conta com a participação de empresas privadas e utiliza soluções de *software* proprietárias como o LabView [10] para representar graficamente, interagir e analisar dados dos dispositivos programáveis.

Esse WebLab foi escolhido no contexto da pesquisa por causa de sua arquitetura formada por módulos funcionais bem definidos e diversas implementações podem seguir a arquitetura para interagir com dispositivos físicos dos mais diferentes tipos. Mas apesar do uso de soluções proprietárias reduzirem o esforço no desenvolvimento do *software* de interação usuário/experimento e entre laboratório/recursos físicos, a disponibilização do WebLab torna-se dependente do sistema operacional e das ferramentas associadas a ele. De forma semelhante, a atualização das versões dos *softwares* proprietários podem impactar no funcionamento do laboratório. O custo das licenças de uso também podem inviabilizar as implementações.

Por outro lado, o WebLab-Deusto [11], que é um laboratório de microeletrônica da Faculdade de Engenharia da Universidade de Deusto, procurou evoluir de soluções de *software* proprietárias para soluções de *software* livre e de código-fonte aberto. No entanto, nem todos os componentes do WebLab seguem essa alternativa: o servidor atualmente conta com um sistema operacional Microsoft Windows e a tecnologia ASP .NET para promover a interação com *Web Services* entre o servidor e as aplicações dos clientes.

As características desse WebLab se assemelham à proposta dessa dissertação ao utilizar soluções de *software* livre e de código-fonte aberto, e também em função de sua arquitetura, que utiliza módulos conhecidos como micro-servidores instalados nos dispositivos lógicos programáveis.

A arquitetura do WebLab-Deusto utiliza uma solução Web baseada em AJAX (*Asynchronous Javascript with XML*) com o uso de micro-servidores que atuam diretamente sobre os dispositivos lógicos programáveis, conhecidos como PLDs (*Programmable Logical Devices*). A proposta de uso de micro-servidores reduz a quantidade de informação de gerenciamento entre os PLDs e o servidor do laboratório, além de otimizar a comunicação entre os PLDs. A arquitetura utiliza uma proteção de *firewall*, é escalável o suficiente para permitir a adição de muitos dispositivos programáveis e suporta o trabalho colaborativo entre muitos grupos. A Fig. 1.3 mostra a arquitetura de interação do *software*.



Fig. 1.3: Arquitetura de Interação do WebLab-Deusto [11].

O projeto do WebLab-Deusto realizou uma avaliação com os estudantes que utilizaram a ferramenta. A avaliação revelou uma boa qualidade geral quanto ao uso do WebLab, mas os resultados da velocidade de interação do experimento com o usuário e a qualidade das imagens da *webcam* utilizada para representar os eventos receberam a menor pontuação.

Na tentativa de solucionar os problemas de tráfego de informações entre diferentes domínios destacam-se os WebLabs que utilizam SOA (*Service-Oriented Architecture*) [12] como alternativa de comunicação entre o usuário e os recursos físicos do laboratório.

Uma implementação que apresenta uma arquitetura SOA semelhante ao contexto do trabalho possibilita a integração de experimentos utilizando uma arquitetura dupla cliente-servidor. Essa arquitetura utiliza *Web Services* como mediadores da comunicação [13]. A Fig. 1.4 ilustra a arquitetura dupla cliente-servidor. A primeira arquitetura cliente-servidor ocorre entre o *browser* e o servidor Web do WebLab. A segunda arquitetura realiza a comunicação entre o WebLab e os *Web Services*.

Esse trabalho se aproxima da arquitetura para o desenvolvimento de experimentos proposta nesta dissertação da seguinte forma:

1. existência de um provedor de serviços que registra a localização remota dos *Web Services* em um Servidor de Registro;
2. o requisitante do serviço busca no Servidor de Registro pelos recursos que ele precisa e os seleciona;
3. o requisitante do serviço descobre no Servidor de Registro a localização dos recursos. Então, esse requisitante envia mensagens SOAP (*Simple Object Access Protocol*) diretamente para a localização (provedor de serviços que registrou os seus *Web Services*). Na localização, os *Web Services* são contactados.

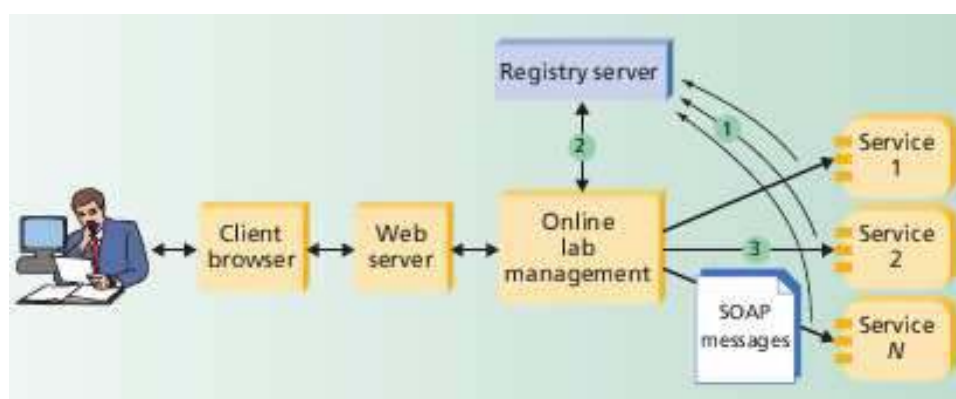


Fig. 1.4: Arquitetura Dupla Cliente-Servidor [13].

Os métodos utilizados no estudo [13] revelaram não ser adequados para o controle de dispositivos em tempo-real devido o baixo desempenho na qualidade da comunicação. A demora ocorre por causa das camadas de transporte adicionais para as mensagens SOAP. O atraso envolve a criação da mensagem em formato SOAP, a associação da mensagem ao protocolo HTTP (*Hypertext Transfer Protocol*), o tempo de transporte na rede e a decodificação da mensagem.

Esse estudo [13] apontou que para um domínio de aprendizagem à distância o atraso na comunicação é compensado pela integração de muitos serviços heterogêneos entre domínios distintos. Nenhuma consideração é feita quanto à segurança de acesso ou segurança na interação com os dispositivos físicos. O trabalho priorizou a descrição de alternativas para o envio de argumentos e parâmetros de controle nas mensagens SOAP.

Outro trabalho selecionado se assemelha com o anterior, mas suporta a integração de muitos serviços heterogêneos com uma arquitetura genérica para WebLabs de 3 camadas, baseada na Web [14]. A Fig. 1.5 ilustra os detalhes dessa arquitetura que fornece modos de acesso compartilhado aos experimentos em redes com baixa largura de banda. Um experimento genérico é modelado com um conjunto de entradas, saídas e regras. Um conjunto de ferramentas para o registro do experimento e do laboratório é sugerido para coletar os metadados para os laboratórios e experimentos com uma necessidade de programação mínima. O intuito é possibilitar rápida integração padronizada de experimentos com o WebLab.

Esse trabalho também foi selecionado por explicar uma abordagem de gerência semelhante à proposta nessa dissertação. Três diferentes regras são habilitadas para os usuários do WebLab: 1) um administrador responsável por toda a administração do servidor do WebLab; 2) administradores de laboratório que são responsáveis pela gerência dos servidores do laboratório e dispositivos dos experimentos; 3) estudantes que podem interagir com os experimentos.

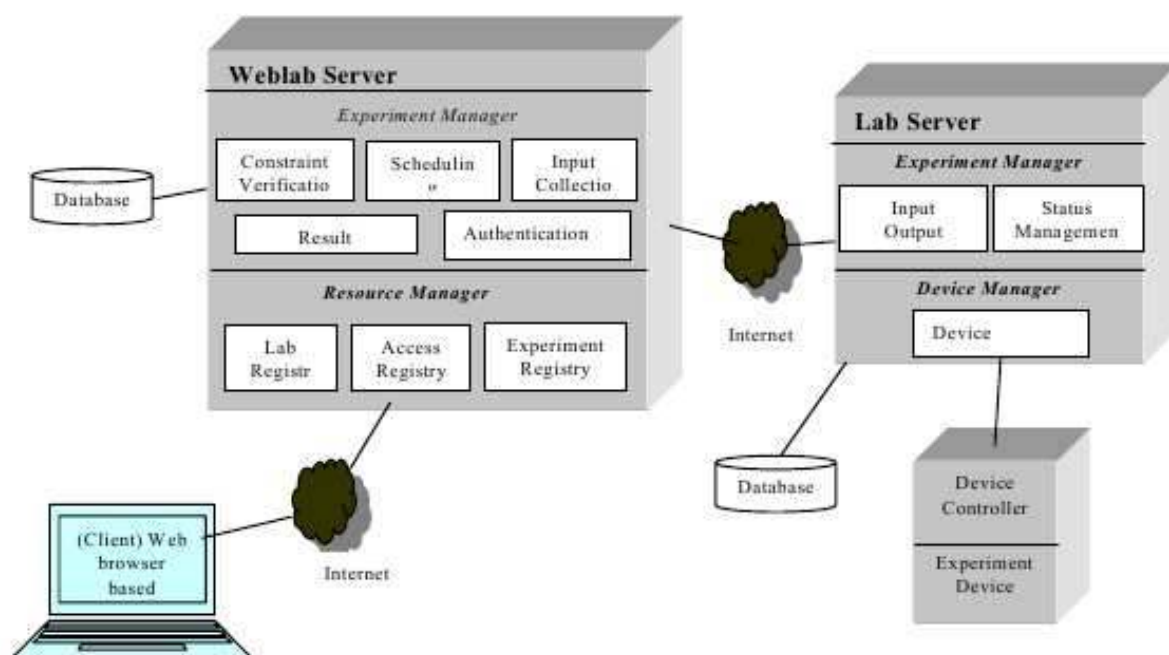


Fig. 1.5: Arquitetura Genérica para um WebLab [14].

A arquitetura separa a camada de adição de novos experimentos do gerenciamento e controle de experimentos e, esses dois últimos, são mantidos por cada laboratório integrante do WebLab. Um experimento é formado por dispositivos que podem ser controlados remotamente. A autorização, autenticação e registro de usuários é centralizada, mas o agendamento de usuários para experimentos é realizado apenas no servidor do laboratório.

O trabalho descreve que devem ser associadas regras aos usuários do WebLab [14]. Dependendo da regra o usuário pode ter diferentes tipos de acesso, por exemplo, interação com o experimento, registro de novo experimento ou criação e gerenciamento de contas, por exemplo. A arquitetura propõe ainda que a interação de clientes com os laboratórios remotos seja realizada por meio do WebLab servidor que, por sua vez, encaminha os dados de entrada para o laboratório remoto correspondente.

A arquitetura prevê que muitos laboratórios possam ser adicionados ao WebLab. Por isso, um registro armazena as informações dos experimentos de cada laboratório. Esse registro inclui as informações de entrada, saída e regras do experimento disponibilizado na Web. Finalmente são definidos dois níveis de segurança para a execução remota: uma no servidor do WebLab e outra no controlador do dispositivo para evitar que entradas incorretas inviabilizem o uso do laboratório.

No entanto, não é definida a arquitetura da aplicação utilizada pelo usuário para acessar os experimentos. A portabilidade é citada no acesso ao laboratório com o uso de um *browser*, mas não são feitas considerações quanto à portabilidade da comunicação. A implementação proposta da arquitetura utiliza *sockets* junto ao controlador do dispositivo remoto para a interação com os dispositivos gerenciáveis e os parâmetros dos experimentos foram mantidos em uma base de dados. As informações do *browser* do usuário devem passar pelo *WebLab Server* e pelo *Lab Server* antes de atingir o dispositivo remoto gerenciável, o que pode prejudicar a *performance* da interação quando muitos usuários de diferentes WebLabs acessarem experimentos ao mesmo tempo. Os WebLabs estudados orientaram a proposta para o desenvolvimento de experimentos nessa dissertação. No entanto, o WebLab proposto segue a arquitetura orientada a serviços do WebLab GigaBOT [6].

O projeto “GigaBOT - Laboratórios de Acesso Remoto sobre Redes Avançadas” tem o objetivo principal de oferecer uma plataforma para suporte a aplicações multimídia em redes de alta velocidade com soluções para gerência integrada de aplicações. O WebLab GigaBOT é um WebLab no domínio da robótica móvel que opera sobre uma rede de alto desempenho, a rede Giga da Rede Nacional de Ensino e Pesquisa (RNP) [15].

O projeto de laboratórios virtuais de robótica teve início em 1996 no Centro de Pesquisas Renato Archer (CenPRA - que atualmente recebe o nome de Centro de Tecnologia da Informação Renato Archer - CTI Renato Archer) com o projeto REAL (*Remotely Accessible Laboratory*) que foi o primeiro laboratório virtual na área de robótica desenvolvido no país [16]. A partir de 1997, um acordo de cooperação entre o CenPRA e a FEEC/UNICAMP teve o objetivo de oferecer uma infraestrutura de código-fonte gratuito e aberto para a construção de WebLabs [17]. Ao longo desses anos, diversas tecnologias foram empregadas para a construção de WebLabs, tais como objetos distribuídos [18], componentes de software [19] [20] [21] [22] e arquiteturas orientadas a serviços [6] [23] [24] [25].

O WebLab GigaBOT foi concluído em 2006 e é a continuação de várias implementações de infraestruturas de WebLabs de robótica do projeto REAL. Esse WebLab realiza o controle de robôs à distância onde os experimentos são formados por meio da composição de serviços [6].

O modelo de referência do projeto GigaBOT permite que diversos WebLabs sejam integrados, como mostra a Fig. 1.6. O diagrama UML (*Unified Modeling Language*) mostra os principais elementos do modelo e os relacionamentos entre eles. Os elementos centrais são o participante, que pode ser um usuário individual ou um grupo, e o WebLab. Para utilizar um WebLab o participante deve possuir credenciais e estabelecer uma ou mais sessões com o WebLab. As credenciais e sessões são exclusivas para os participantes que acessam um WebLab específico. As credenciais são os papéis (estudante, instrutor, por exemplo) e as permissões (uso, gerenciamento, por exemplo) atribuídos ao participante.

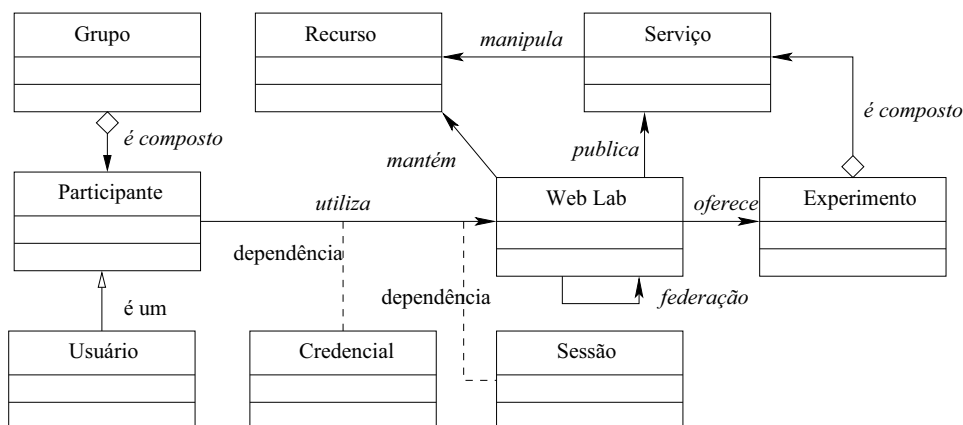


Fig. 1.6: Modelo de Referência para WebLabs [6].

Uma sessão gerencia a interação entre o participante e o WebLab, como o tempo restante e operações de *log*. Cada WebLab disponibiliza os serviços que ele oferece e esses serviços são unidades que realizam atividades bem definidas, tais como acesso, interação, comunicação e gerência, respeitando a proposta da arquitetura SOA. Os experimentos oferecidos por cada WebLab possuem um conjunto de recursos físicos tais como interfaces de rede e câmeras, e lógicos, tais como configuração de IP, rotas, e assim por diante. Os recursos de cada experimento são manipulados remotamente por meio de serviços. Finalmente, um WebLab pode interagir com outros WebLabs.

1.2.2 Implementação da Arquitetura *DiffServ*

Diversas abordagens foram sugeridas para contornar o problema da reserva de recursos na Internet. Dentre elas, destaca-se o uso de um componente centralizador de requisições de parâmetros de QoS, distribuidor e gerenciador de recursos conhecido como *Bandwidth Broker* (BB) [5].

Dentre os trabalhos selecionados a implementação de um domínio *DiffServ* seguindo uma arquitetura multi-camadas junto a um ISP [26] se assemelha ao trabalho dessa dissertação ao apresentar um BB que interage com elementos presentes em outras camadas lógicas no domínio. Além disso, as extensões mostram que a arquitetura *DiffServ* permite a adição de novos elementos funcionais sem comprometer os elementos que atuam diretamente no domínio *DiffServ*. No entanto, não são apresentadas soluções quanto à portabilidade da adição dos novos elementos em outros domínios.

A arquitetura multi-camadas utiliza o BB e é formada por uma camada de controle de recursos que é dividida em duas sub-camadas: a camada superior é responsável pelo controle administrativo da rede e a camada inferior realiza a política de controle de admissão dos fluxos.

O modelo assume que a Internet está separada em vários domínios administrativos ou ISPs. Cada componente interno da rede realiza o encaminhamento com base no modelo *DiffServ* de agregação de fluxos, respeitando as políticas de ingresso/egresso no domínio, bem como o SLA (*Service Level Agreement*) estabelecido entre os vários domínios. O BB é o componente responsável por oferecer de forma centralizada os parâmetros de QoS, monitorando e controlando esses parâmetros no domínio. Para isso, a sincronização entre os nós pode ser feita utilizando o modelo IntServ/RSVP, CORBA, entre outros. Uma Camada de Controle de Recursos (*Resource Control Layer* - RCL) é adicionada ao domínio, como mostra a Fig. 1.7.

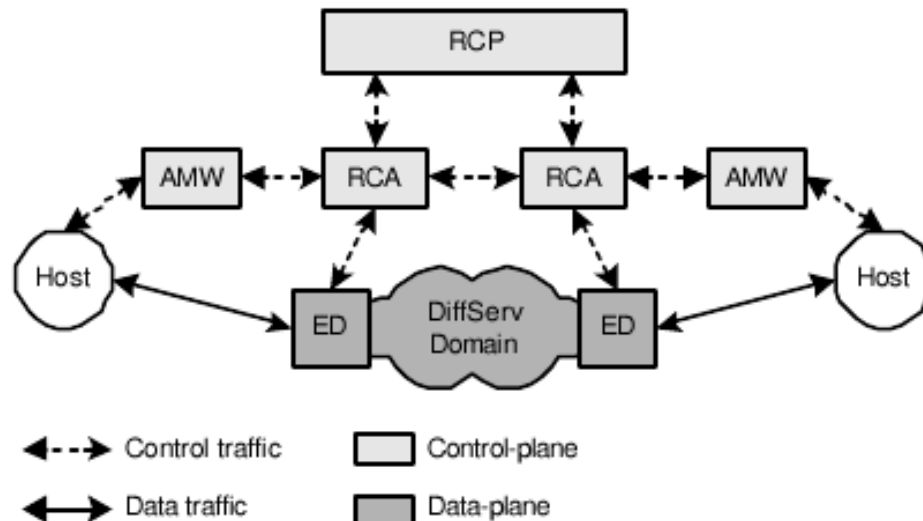


Fig. 1.7: Arquitetura Multi-Camadas para um Domínio *DiffServ* [26].

Essa camada é dividida em três componentes lógicos: a) um Ponto de Controle de Recursos (*Resource Control Point* - RCP) responsável pelo gerenciamento e distribuição de recursos da rede; b) em cada nó existe um Agente de Controle de Recursos (*Resource Control Agent* - RCA) responsável por policiar a admissão de controle. Isso é necessário para verificar se a quantidade de recursos disponíveis é suficiente para o uso do cliente. Esse policiamento ocorre nos dispositivos de borda do domínio; c) uma Aplicação de *Middleware* (*Application Middleware* (AMW)) que fornece uma interface para o usuário final sinalizar os requerimentos de QoS que o domínio deveria oferecer.

mediador da comunicação. O trabalho contribui ao apresentar exemplos de *scripts* da configuração *DiffServ*, mas não são descritas a arquitetura e os detalhes da implementação do BB. A figura Fig. 1.9 ilustra a arquitetura desse sistema.

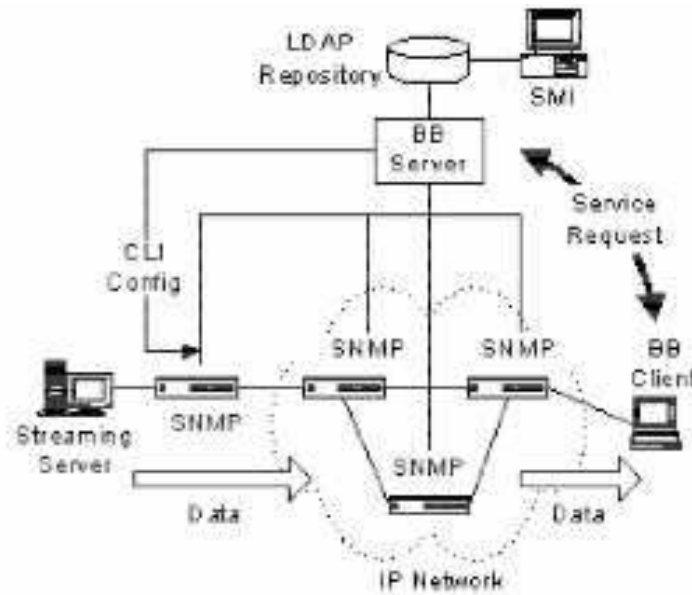


Fig. 1.9: Arquitetura de um BB com Suporte a QoS e SNMP [31].

1.3 Contribuições

O NetLab WebLab atende às necessidades da experimentação remota na área de redes e de Serviços Diferenciados. Desde a instalação física e lógica até a interação do usuário com o laboratório foram almejadas soluções gratuitas com preocupação especial para a portabilidade. O domínio *DiffServ* pode ser acessado em diferentes sistemas operacionais com o uso de um aplicativo Java acessível com um Web browser. Apenas uma versão recente de JRE (*Java Runtime Environment*) precisa ser instalada no *host* do usuário para acessar os experimentos. Priorizou-se também o desempenho da comunicação remota, o desempenho da comunicação interna entre os recursos do laboratório, a segurança da execução e a simplificação das políticas de transferência de dados entre *firewalls* e *proxies* de diferentes domínios.

As contribuições seguem com a proposta de extensão à arquitetura *DiffServ* para melhor atender aos requisitos de *QoS*. Complementam as funções do BB proposto os módulos para a negociação de recursos intra-domínio e para a realização das configurações *DiffServ* nos *hosts* do domínio.

Além disso, o *software* dos experimentos foi desenvolvido independente das aplicações de gerência e uso do WebLab. Isso possibilita que os experimentos sejam mantidos com redução do espalhamento de código. O código-fonte que implementa um experimento está contido apenas no projeto deste experimento. O padrão de projetos adotado permite que novos experimentos sejam criados com reduzida necessidade de codificação. Foram realizadas alterações na gerência de uso dos experimentos e nos serviços de interação do projeto GigaBOT para melhorar o acesso aos experimentos e o suporte à escalabilidade de representação de recursos.

O padrão do projeto permitiu ainda a gerência distribuída das configurações do domínio *DiffServ* com a confecção do experimento administrativo BB. O experimento centraliza as configurações *DiffServ* do domínio, abstrai e simplifica as complexas configurações da tecnologia com o uso de

softwares gráficos. Os demais experimentos *DiffServ* são capazes de recolher as informações do experimento anterior e respeitar as restrições para o tráfego submetido à rede interna.

Esse padrão de projeto também orientou a criação de uma arquitetura com suporte à composição de serviços, ou seja, novos experimentos podem ser gerados a partir da agregação de serviços que realizam tarefas específicas.

Em resumo, buscou-se atingir os seguintes objetivos:

- propor uma infraestrutura física e lógica em um laboratório de redes de computadores para permitir a realização de experimentos remotos em um domínio de serviços diferenciados;
- propor a realização de diversos experimentos remotos na área de redes de computadores. Estes experimentos devem atuar diretamente sobre os recursos físicos do laboratório;
- permitir o acesso aos recursos do laboratório remoto com a interação entre aplicativos que utilizam a tecnologia de *Web Services*;
- utilizar serviços de acesso ao laboratório para permitir o controle administrativo (cadastro de usuários, experimentos e recursos, disponibilização de experimentos, entre outros) e o controle do agendamento de uso dos experimentos do laboratório pelo próprio usuário;
- explicar os detalhes que levaram à escolha da infraestrutura do laboratório: características da comunicação entre redes distintas, comunicação interna entre os *hosts* do domínio, paradigma da programação orientada a serviços, composição de serviços e computação distribuída;
- estender a arquitetura *DiffServ* proposta pelo IETF (*Internet Engineering Task Force*) com a finalidade de oferecer a negociação global dinâmica de recursos intra-domínio com o uso de um BB e monitorar a qualidade do serviço garantido por meio de contrato estabelecido com os usuários de experimentos *DiffServ* no laboratório;
- propor experimentos *DiffServ* didáticos no laboratório de redes.

1.4 Contexto do Trabalho

O trabalho apresentado nesta dissertação participou do projeto GigaBOT [33] que é uma nova abordagem para a criação de WebLabs. Os objetivos principais do projeto GigaBOT são o suporte a aplicações multimídia de tempo-real em redes avançadas, um WebLab para robótica móvel e gerência integrada de redes e outras aplicações.

Atualmente, os resultados para a elaboração dessa dissertação permitiram o ingresso no projeto KyaTera [34] que conta com redes ópticas voltadas para o estudo da Internet do futuro. O projeto conta com um ambiente colaborativo aberto para a participação de grupos de pesquisas que tenham interesse em contribuir com inovações tecnológicas e conhecimento científico na rede KyaTera.

1.5 Estrutura da Dissertação

O conteúdo da dissertação foi dividido em seis capítulos. O Capítulo 2 descreve os requisitos funcionais para a criação de um WebLab de redes com suporte à Arquitetura Orientada a Serviços utilizando a Computação Orientada a Serviços (*Service Oriented Computing* - SOC) para implementar e compor os blocos funcionais da arquitetura.

O Capítulo 3 apresenta a arquitetura *DiffServ* e os requisitos funcionais de um WebLab de redes com suporte a experimentos *DiffServ*.

As arquiteturas para a criação e gerência do WebLab assim como a criação de experimentos a partir da composição de serviços são apresentados no Capítulo 4.

O Capítulo 5 discute a implementação do WebLab e de seus experimentos seguindo as arquiteturas propostas no capítulo anterior, com a junção dos requisitos funcionais de WebLabs SOA com suporte a *DiffServ*. Os resultados obtidos justificam a metodologia para o desenvolvimento de novos experimentos no WebLab.

Finalmente, o Capítulo 6 tece as considerações finais do trabalho.

Capítulo 2

WebLabs de Redes com Suporte a SOA

Este capítulo descreve os requisitos funcionais que devem ser seguidos para a implementação de WebLabs de Redes de Computadores baseados na arquitetura SOA. Os requisitos funcionais de redes estão relacionados às atividades e recursos da experimentação e, por isso, são específicos para esse tipo de laboratório. Por outro lado, a descrição da arquitetura SOA e das tecnologias que seguem o seu padrão orientam a definição dos requisitos funcionais genéricos necessários para a interação externa e uso eficiente de experimentos em WebLabs. A arquitetura SOA auxilia na implementação e configuração do WebLab ao utilizar *Web Services* como ferramentas para a integração das partes distintas que compõe a infraestrutura de *software* do laboratório.

2.1 Requisitos Funcionais para WebLabs de Redes

WebLabs podem ser classificados pelo tipo de controle remoto que eles exercem sobre os recursos dos experimentos [8]. Para um WebLab de Redes de Computadores ser utilizado como ferramenta didática ele precisa oferecer um alto nível de interação com os dispositivos físicos e ser capaz de alterar a configuração lógica da rede. A atuação não deve estar restrita à atribuição de parâmetros nos dispositivos físicos porque a lógica da interação e as características de cada um desses recursos influencia na qualidade da comunicação entre os elementos do laboratório. Nesse contexto, a presença de interfaces gráficas amigáveis oferece um atrativo à experimentação, reduz a necessidade do conhecimento prévio da lógica de comunicação, auxilia na percepção de como essa lógica influencia na qualidade da comunicação, simplifica a visão geral da rede e amplia as possibilidades de interação com os recursos do laboratório.

Para o uso adequado desse tipo de WebLab é necessária a geração de manuais de consulta que orientem o usuário nos experimentos. A padronização do formato desses manuais deve ser considerada importante para simplificar a consulta ao longo dos diversos experimentos. Como sugestão, a documentação poderia ser semiestruturada em formato XML e apresentar as seguintes seções: nome do experimento, resumo descritivo, funcionamento, exemplos de uso, *bugs* conhecidos, autores e informações adicionais.

Um WebLab de Redes de Computadores deve ser capaz de gerenciar diversos dispositivos físicos e *softwares* de interação. A presença de *softwares* de gerenciamento simplifica a adição de novos recursos físicos. Também é necessária a gerência dos recursos lógicos (*software*) que atuam sobre os dispositivos físicos. Esses recursos lógicos podem realizar desde a coleta filtrada de informações à simulação de protocolos de transporte. Por isso, o desenvolvimento de experimentos depende da infraestrutura física e lógica da rede para que diversos usuários possam acessar esses recursos apenas nos setores definidos na experimentação, sem causar alterações em outras partes da rede. A gerência

dos recursos físicos e lógicos possibilita a criação de novos experimentos utilizando combinações adequadas desses recursos.

Esses laboratórios devem oferecer soluções seguras para a interação com os dispositivos físicos e os *softwares* de interação com a rede. Para isso, é necessário uma rede de retaguarda que faça a reinicialização dos recursos em um estado estável, de forma que não sejam perdidas as conexões entre os elementos em caso de erros ou excesso de utilização. Um WebLab de redes também deve ser capaz de avaliar o resultado das configurações submetidas aos experimentos. Isso pode ser feito com ferramentas de medição do tráfego porque elas auxiliam no entendimento prático dos fatores que influenciam o desempenho da comunicação.

Por outro lado, a disponibilidade desse tipo WebLab será dependente da manutenção de diversos recursos físicos e, por isso, a utilização do WebLab nos 7 dias da semana, 24 horas por dia (24x7) não é um requisito funcional essencial, mas sim a qualidade com que os experimentos são ofertados. Essa qualidade diz respeito à comunicação, tempo de resposta, preparação remota adequada, disponibilização adequada dos recursos, entre outros, que influenciam na precisão do resultado final.

Diversas alternativas de simulação em redes são focadas na análise do comportamento do tráfego em diferentes topologias, recursos e qualidades de enlace [28] [29] [30]. As mesmas atividades podem ser realizadas com a instrumentação remota “real” com resultados mais precisos, mas é necessário a oferta de QoS no enlace entre o usuário e o laboratório.

Os WebLabs de redes também são ferramentas úteis para a análise do fluxo de pacotes para aplicações multimídia e de tempo-real. Essas aplicações são mais exigentes quanto às características do enlace e, por isso, o uso de *brokers* para realizar a reserva e controle de recursos de forma centralizada e dinâmica, independente da interação direta com o usuário, simplifica a experimentação.

Como opção, o laboratório poderia fornecer alternativas de desenvolvimento de pacotes de *software* padronizados que complementem a experimentação no ambiente de testes. Os elementos de *software* potencializam o uso do laboratório ao viabilizar o estudo colaborativo de soluções de comunicação que dependem de algoritmos bem definidos, como exemplo, diferentes implementações de protocolos de redes. Os resultados obtidos com as medições locais na submissão de fluxos auxiliam no entendimento de como os diferentes algoritmos influenciam o desempenho do tráfego na rede. Em virtude da necessidade do reaproveitamento e interoperabilidade entre os componentes de *software* com interfaces bem definidas, a criação de novos experimentos e a adição de novas funcionalidades deve ser realizada com reduzida necessidade de codificação. Em virtude disso, SOA oferece boas soluções para a interação entre os componentes de *software* do WebLab de forma não invasiva.

2.2 Visão Geral da Arquitetura SOA

A arquitetura SOA é baseada em um modelo cliente/servidor em que o cliente faz o papel de requisitante de serviços e o servidor de provedor de serviços. A comunicação é baseada na troca de mensagens síncronas ou assíncronas independentes do protocolo de transporte utilizado [35]. A Fig. 2.1 ilustra uma visão geral simplificada da arquitetura SOA.

A implementação de serviços pode ser feita utilizando *Web Services*, mas essa utilização não é obrigatória porque as funcionalidades podem ser implementadas independente da linguagem de programação ou do sistema operacional. Na arquitetura SOA as aplicações distribuídas utilizam os serviços como blocos de construção [36]. BPEL4WS (*Business Process Execution Language for Web Services*) é uma linguagem promissora na padronização da composição de *Web Services* em processos de negócio [37], mas essa composição pode ser realizada na própria codificação da aplicação.

Para simplificar a localização desses serviços em provedores distintos, SOA integra um componente para o registro e a descoberta de serviços. Com isso, os provedores registram os seus serviços

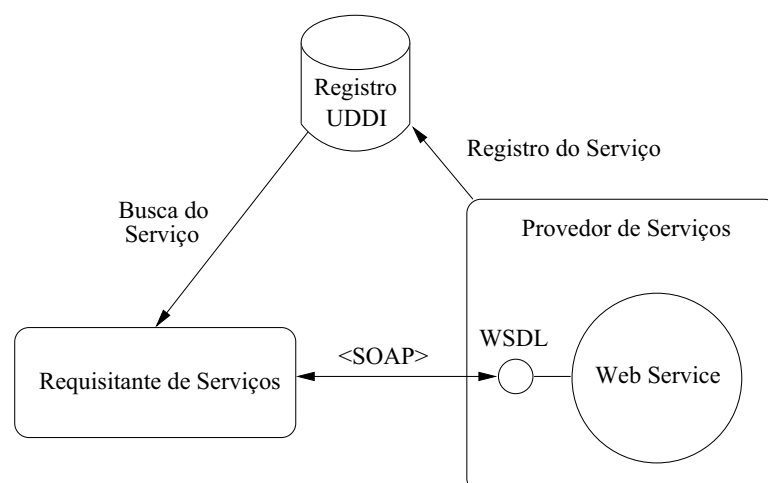


Fig. 2.1: Visão Geral Simplificada da Arquitetura SOA.

em um repositório centralizado. O registro descreve as informações de cada serviço e a localização do provedor que os mantêm. Quando o cliente deseja utilizar um serviço, ele faz uma consulta nesse repositório e informa quais os requisitos que o serviço buscado deve ter. O repositório então devolve uma lista de serviços que satisfazem os requisitos solicitados. Após a escolha do serviço, o cliente acessa diretamente o provedor de serviços porque a lista do repositório traz as informações sobre os parâmetros e a localização do provedor que possui o serviço que satisfaz a funcionalidade solicitada. Essa localização dinâmica pode ser feita no momento da execução do aplicativo. UDDI (*Universal Description, Discovery, and Integration*) especifica um método padrão para publicação e descoberta de componentes de *software* na Web segundo a arquitetura SOA [38].

As unidades de *software* são desenvolvidas independentes umas das outras, encapsulam a lógica da implementação do serviço e expõe apenas a interface de acesso à funcionalidade. Isso torna possível o desenvolvimento de aplicações distribuídas com fraco acoplamento. Além disso, a possibilidade de reuso de código e de disponibilidade na Web permite a confecção de extensas aplicações em ambientes colaborativos distintos.

2.2.1 Web Services

Um *Web Service* é um sistema de *software* para a interação de *hosts* em uma rede. Cada *Web Service* possui uma URI (*Universal Resource Identifier*) que o identifica, uma interface de acesso em formato XML (*Extensible Markup Language*) e suporta a interação direta com outros agentes de *software* utilizando um protocolo de troca de mensagens XML juntamente com protocolos utilizados na Internet [39].

Dentre as motivações relevantes para o desenvolvimento de aplicações distribuídas utilizando *Web Services* está a flexibilidade na utilização desses componentes. Eles podem ser instanciados em diferentes *hosts* sem a necessidade de alterar o seu espaço de endereçamento único (URI) que o identifica no domínio do *host*. Outra motivação está na dificuldade inerente de promover a comunicação entre aplicações remotamente distribuídas diante da grande diversidade de políticas de tratamento de dados em *proxies* e *firewalls* de redes distintas que impedem a transferência imediata de dados de origem duvidosa. A política de tratamento dessas informações utilizando *Web Services* se dá de forma mais simples, utilizando protocolos bem definidos da camada de aplicação do modelo Internet como SMTP e HTTP/HTTPS, por exemplo, como protocolos de transporte. Isso permite que mensagens XML trafeguem entre *proxies* e *firewalls* sem sofrerem restrições, à priori.

Essa característica não ocorre em implementações Java RMI (*Remote Method Invocation*) [40], CORBA (*Common Object Request Broker Architecture*) [41] e DCOM (*Distributed Component Object Model*) [42], por exemplo, que utilizam RPC (*Remote Procedure Call*). No entanto, em comparação com essas tecnologias, o transporte de grandes mensagens XML implica em uma perda de desempenho maior no tráfego de informações entre sistemas distribuídos.

2.2.2 SOAP

O protocolo SOAP (*Simple Object Access Protocol*) codifica e define o formato XML das mensagens transmitidas na rede. Esse protocolo também é utilizado para acessar as funções do *Web Service* e definir o uso de protocolos da camada de aplicação (geralmente SMTP e HTTP) como protocolos de transporte [43] [44].

O protocolo SOAP utiliza a tecnologia XML para definir um *framework* para a construção de mensagens independentes de linguagem de programação ou de semântica proprietária. Com o objetivo de oferecer simplicidade e portabilidade, características como confiabilidade, segurança e roteamento, por exemplo, foram deixadas como extensões ao protocolo [43].

Uma questão relevante é a preocupação com a segurança das informações transmitidas. A implementação do protocolo SOAP conhecida Apache/Axis2 utiliza o módulo Rampart para oferecer mecanismos de segurança na troca de mensagens. Esse módulo permite a encriptação, adição de *timestamp* e de assinatura digital às mensagens XML enviadas [45].

Uma mensagem SOAP é um documento XML que contém:

- um elemento *Envelope* que identifica o documento XML como uma mensagem SOAP;
- um elemento *Cabeçalho* opcional com informações sobre autenticação, roteamento ou quais nós intermediários podem acessar as mensagens;
- um elemento *Corpo* com as informações para enviar e receber mensagens;
- um elemento *Falta* opcional para informar sobre erros no processamento da mensagem.

Segue abaixo o esqueleto de uma mensagem SOAP:

```
1 <?xml version="1.0"?>
2   <soap:Envelope
3       xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
4       soap:encodingStyle=
5           "http://www.w3.org/2001/12/soap-encoding">
6
7       <soap:Header>
8         ...
9       </soap:Header>
10
11      <soap:Body>
12        ...
13        <soap:Fault>
14          ...
15        </soap:Fault>
16      </soap:Body>
17    </soap:Envelope>
```

2.2.3 WSDL

Atualmente, a tecnologia de *Web Services* procura estar em conformidade com a arquitetura SOA ao suportar a interação através da rede e expor uma interface padronizada (*Web Service Description Language* - *WSDL*) que permite o acesso às funcionalidades do serviço [46]. Os *Web Services* são utilizados como aplicações distribuídas capazes de se comunicar umas com as outras independente da plataforma ou linguagem de programação utilizada. Um *Web Service* possui uma interface em formato XML que descreve as funções do *Web Service*, os tipos de dados que ele utiliza e a localização do serviço. Essa interface é formada utilizando as regras da linguagem WSDL [36].

Para ilustrar como o documento WSDL é formado, dado o endereço `www.***.com.br`, o *Web Service* com o nome *BB* (*Bandwidth Broker*) pode ser definido em `axis2/services/BB`. Então, o endereço `www.***.com.br/axis2/services/BB` é chamado de *endpoint* do *Web Service* [47].

Um *Web Service* pode ter uma ou mais operações. Para que a operação seja única na Internet, ela é definida no espaço de nomes (*namespace*) do *host* que mantém o endereço `www.***.com.br`. Um *namespace* é semelhante a um pacote Java, mas formado utilizando a sintaxe URL (*Uniform Resource Locator*). Por exemplo, a operação `addBB_BD` pode ser definida dentro do *namespace* `http://bb.ws`. No entanto, isso não significa que o *namespace* estará acessível via *browser*. Na verdade, ele serve apenas como um identificador único a partir do qual as operações são definidas. Dessa forma, um *namespace* e suas operações podem ser movidas para diferentes *endpoints* sem necessidade de alteração absoluta do caminho do serviço. O *Web Service* é acessado por meio de suas interfaces. A Fig. 2.2 ilustra a estrutura geral da interface do *Web Service*.

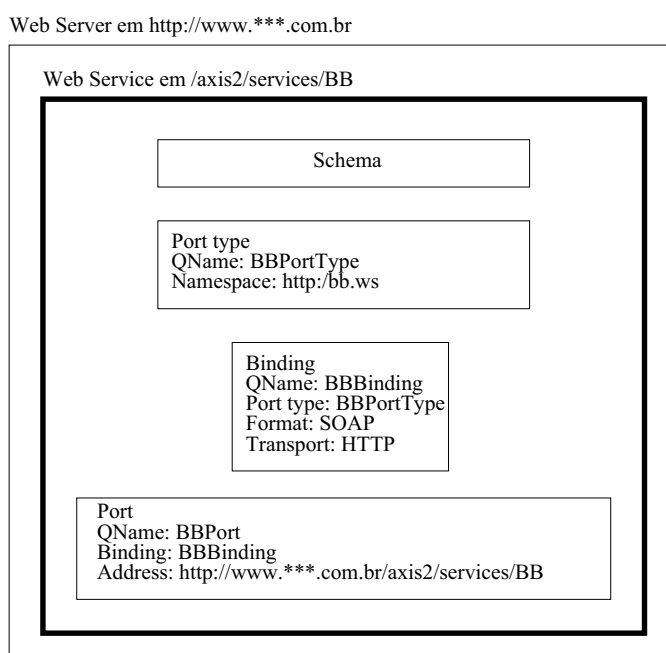


Fig. 2.2: Estrutura Geral da Interface do *Web Service* BB.

QName

Um nome qualificado (*QName*) é um nome sujeito a interpretação no *namespace* [48]. Como exemplo, o nome `addBB_BD` é um *QName* no *namespace* `http://bb.ws`. Da mesma forma, `string` é um *QName* no *namespace* `http://www.w3.org/2001/XMLSchema`. Em *Web Services*, a chamada de um

método (ou chamada de uma operação) é conhecida como “mensagem de entrada” e os parâmetros como “partes”. O valor retornado é chamado “mensagem de saída” [47].

Schema

Um *schema* define as regras para compor e validar a estrutura de documentos XML [49]. Com o *schema* é possível simplificar a representação do serviço ao agrupar as definições das regras de composição do documento XML no *namespace* informado no início da mensagem, seja ela uma mensagem de entrada ou de saída.

Port type

As operações em *Web Services* são agrupadas em *port types* [47]. Um *port type* é definido com um *QName*, ou seja, um nome local que pode ser interpretado no *namespace*.

Binding

Um *port type* pode ser acessado utilizando diferentes protocolos de transporte. Por exemplo, uma mensagem pode ser transportada em uma requisição HTTP POST ou em um e-mail (SMTP). Cada uma dessas associações é conhecida como *binding* [47].

Port

Um *port* define a “porta” de entrada para acessar o *Web Service*. No *port* é definido o endereço através do qual o *Web Service* pode ser acessado. Cada *port* referencia um *binding* [47]. Podem ser utilizados diferentes *ports* para receber requisições de diversos *hosts*. No entanto, cada *host* pode apontar para um *binding* diferente. O objetivo de se utilizar *ports* é permitir a distribuição das formas de acesso aos *Web Services*. Os *ports* referenciam *bindings* que, por sua vez, referenciam *port types*. Dessa forma, diferentes *ports* que apontam para o mesmo *binding* suportam as mesmas operações.

Namespace

Um *namespace* em particular é chamado de *target namespace* [50]. Há um único *namespace* para que o *Web Service* defina suas operações dentro dele. Nesse caso, o *namespace* e o *target namespace* são equivalentes dentro do mesmo *Web Service*. O acesso às operações de *Web Services* pode ser realizado utilizando uma URI que informa a localização do *namespace*. Uma URI pode ser de dois tipos: uma URL ou uma URN (*Uniform Resource Name*), que é uma URI que utiliza um schema.

Em resumo, um *Web Service* tem um ou mais *ports* que referenciam *bindings*. Um *binding* referencia um *port type* e informa qual o protocolo que será utilizado no transporte das mensagens. O *port type* contém uma ou mais operações formadas para mensagens de entrada e de saída. Cada mensagem repete as regras impostas pelo *schema* do *Web Service*. Um *Web Service* possui um *QName* único que o identifica. Da mesma forma, cada *port*, *binding*, *port type* e operação possui um *QName* único.

O trecho a seguir ilustra o documento WSDL para o *Web Service Bandwidth Broker*:

```
1 <?xml version="1.0" encoding="UTF-8"?>
  ...
3   <wsdl:types>
4     <xs:schema xmlns:xsd="http://bb.ws"
      attributeFormDefault="qualified"
      elementFormDefault="qualified"
      targetNamespace="http://bb.ws">
```

```

5          <xs:element name="addBB_BD">
6              <xs:complexType>
7                  <xs:sequence>
8                      <xs:element minOccurs="0"
                          name="hostRetaguarda"
                          nillable="true"
                          type="xs:string"/>
...
14          </xs:sequence>
15      </xs:complexType>
16  </xs:element>
...
1342 </xs:schema>
1343 </wsdl:types>
...
1356 <wsdl:message name="addBB_BDRequest">
1357     <wsdl:part name="parameters"
                  element="ns0:addBB_BD"/>
1358 </wsdl:message>
1359 <wsdl:message name="addBB_BDResponse">
1360     <wsdl:part name="parameters"
                  element="ns0:addBB_BDResponse"/>
1361 </wsdl:message>
...
1752 <wsdl:portType name="BBPortType">
...
1761     <wsdl:operation name="addBB_BD">
1762         <wsdl:input message="ns0:addBB_BDRequest"
                      wsaw:Action="urn:addBB_BD"/>
1763         <wsdl:output message="ns0:addBB_BDResponse"
                       wsaw:Action="urn:addBB_BDResponse"/>
1764     </wsdl:operation>
...
2025 </wsdl:portType>
2026 <wsdl:binding name="BBSOAP11Binding" type="ns0:BBPortType">
2027     <soap:binding transport=
        "http://schemas.xmlsoap.org/soap/http"
        style="document"/>
...
2046     <wsdl:operation name="addBB_BD">
2047         <soap:operation soapAction="urn:addBB_BD"
                          style="document"/>
2048         <wsdl:input>
2049             <soap:body use="literal"/>
2050         </wsdl:input>
2051         <wsdl:output>
2052             <soap:body use="literal"/>
2053         </wsdl:output>
...
3870 </wsdl:binding>
3871 <wsdl:service name="BB">
3872     <wsdl:port name="BBSOAP11port_http"
                  binding="ns0:BBSOAP11Binding">
3873         <soap:address location=
            "http://localhost:8080/axis2/services/BB"/>
3874     </wsdl:port>
3875     <wsdl:port name="BBSOAP12port_http"
                  binding="ns0:BBSOAP12Binding">

```

```
3876         <soap12:address location=  
3877             "http://localhost:8080/axis2/services/BB"/>  
3878     </wsdl:port>  
3879     <wsdl:port name="BBHttpport"  
3880         binding="ns0:BBHttpBinding">  
3881         <http:address location=  
3882             "http://localhost:8080/axis2/services/BB"/>  
3883     </wsdl:port>  
3884 </wsdl:service>  
3885 </wsdl:definitions>
```

2.3 Requisitos de um WebLab de Redes com Suporte SOA

O paradigma da Computação Orientada a Serviços (SOC) segue a arquitetura SOA para o desenvolvimento de aplicações distribuídas que utilizam serviços como blocos de construção. Para um laboratório de redes o paradigma oferece alternativas para a comunicação entre os usuários e os recursos do laboratório.

O servidor do WebLab deve manter as informações dos serviços que possui e fornecer o acesso externo aos experimentos que interagem com os serviços. Ambos são possíveis com a instanciação de um *container* SOA capaz de manter e listar as interfaces dos serviços, e intermediar a comunicação entre o domínio do usuário e o WebLab. Requisitos como o agendamento do uso do laboratório e a segurança de acesso ao WebLab, por exemplo, podem ser implementados como serviços, mas a arquitetura SOA não prevê como deve ser feito o gerenciamento de conexões, a segurança e a coordenação entre eles. O uso de um servidor de aplicações Web é uma opção para disponibilizar e centralizar as aplicações de interação, acesso e gerenciamento.

Para evitar inconsistências, um laboratório de redes deve planejar a distribuição de seus experimentos para que um mesmo recurso não seja configurado por mais de um usuário ao mesmo tempo. A SOC auxilia nesse processo com o fraco acoplamento das associações entre recursos e experimentos reduzindo o esforço da manutenção, mas o administrador do WebLab precisa gerenciar as composições para verificar as inconsistências.

O laboratório deve possuir serviços específicos e independentes para a interação com cada recurso físico ou lógico. Com isso, novos experimentos são criados com reduzida necessidade de codificação com a composição de serviços. O uso de um serviço compartilhado entre os experimentos reduz o esforço para a atualização das funcionalidades dos experimentos. Outra consequência é o controle de erros e falhas porque o mal funcionamento de um serviço não impede o uso de um experimento, uma vez que existe um fraco acoplamento entre o serviço e o experimento.

O WebLab precisa manter um repositório centralizado capaz de listar e expor as interfaces de seus serviços. O uso de um repositório que siga o padrão UDDI é opcional porque os experimentos irão utilizar, a priori, apenas os serviços do domínio do laboratório, ainda que a interação com serviços de outros domínios seja possível.

A *performance* do tráfego de mensagens XML é dependente da quantidade de informação inserida nos documentos SOAP e das soluções de compactação dessas mensagens [13], além das características inerentes do enlace entre a aplicação cliente e o *Web Service* do WebLab. Por isso são viáveis alternativas como REST [51], para otimizar o uso da largura de banda entre a aplicação cliente e o *Web Service* e reduzir o tempo de resposta de sucessivas interações remotas.

Transferência de Estado Representacional (*Representational State Transfer - REST*) é o termo utilizado para descrever um estilo de arquitetura de sistemas de rede [52]. REST é uma coleção de princípios de rede que descreve como recursos são definidos e endereçados. As implementações que seguem o princípio REST são conhecidas como RESTful e podem utilizar interfaces que transmitem

dados utilizando o protocolo HTTP sem quaisquer camadas adicionais de mensagem tais como SOAP. Mas a definição REST vai além e é possível utilizar sistemas de comunicação sem utilizar HTTP ou mesmo sem interagir com a Internet [51].

A explicação do termo pode ser dada com o seguinte exemplo: um cliente referencia um recurso na Web utilizando uma URL. Uma representação do recurso pode ser a página HTML da URL. Essa representação mantém o cliente em um estado. Quando o cliente acessa um *link* dentro da página, a nova representação transfere o cliente para outro estado [53].

REST não é um padrão e por isso não define o uso de HTTP, URL, representações de recursos em XML, HTML, PNG (*Portable Network Graphics*), entre outros. Para *Web Services* o uso de REST mantém a descrição dos recursos expostos mais simples e permite o controle da forma como esses recursos são acessados utilizando o protocolo da camada de aplicação HTTP, por exemplo. As aplicações que utilizam SOAP enviam arquivos XML em comunicações HTTP POST. Sistemas RESTful têm um controle maior na comunicação, permitindo operações de GET, POST, PUT e DELETE. Ao contrário de SOAP, a URL da requisição REST contém toda a informação necessária para ter acesso ao recurso. Isso implica em menor consumo de banda para requisições quando não for necessária a preparação de uma mensagem XML para ser enviada e decodificada no servidor (HTTP GET, por exemplo), e mesmo no envio de mensagens XML porque não são necessários as informações do envelope ou corpo SOAP. Para descobrir qual recurso uma mensagem SOAP deseja acessar é necessário que o envelope XML seja decodificado no servidor, mas em uma requisição REST é suficiente a informação da URL e protocolo de acesso.

A arquitetura REST é baseada na manutenção de um conteúdo bem formado, ou seja, uma requisição informa qual recurso se quer acessar por meio de uma URL. A mensagem de resposta contém um conjunto de *links* que permitem ao cliente navegar pelos recursos remotos disponíveis, respeitando um modelo de máquina virtual de estados. Essa característica permite analisar a integridade dos relacionamentos oferecidos. No entanto, na implementação SOAP, as mensagens XML são encapsuladas. O servidor que as recebe é obrigado a extrair o conteúdo das mensagens antes de continuar o processo de encaminhamento. As mensagens de resposta não possuem *links* que permitem ao usuário navegar entre os estados remotos disponíveis, ficando a cargo do receptor o entendimento de como processar o conteúdo recebido. Essa característica reduz a escalabilidade e a portabilidade do protocolo SOAP. Tanto os serviços REST quanto os serviços SOAP são descritos utilizando a linguagem WSDL.

Finalmente, REST *Web Services* são um subconjunto da pilha de Web Services na implementação RESTful Axis2/Java [54]. As requisições são por padrão síncronas, os parâmetros são informados na própria URL e as operações HTTP POST não precisam de um envelope ou de um corpo SOAP. As mensagens XML não possuem cabeçalho e apenas o *payload* é enviado.

2.4 Considerações Finais

Os WebLabs de Redes de Computadores baseados na arquitetura SOA oferecem flexibilidade para o desenvolvimento de aplicações distribuídas. A oferta de controle de QoS para os experimentos será explorada no próximo capítulo.

Capítulo 3

WebLabs de Redes com Suporte a *DiffServ*

Este capítulo faz uma descrição dos requisitos que um WebLab de redes de computadores deve possuir para dar suporte a experimentos *DiffServ*. O objetivo principal é permitir o controle do compartilhamento da largura de banda da rede. Esse controle pode ser feito pelos usuários com marcações nos cabeçalhos dos pacotes IP ou por meio de agentes que alocam a banda necessária para o fluxo de acordo com as políticas acordadas para a realização do experimento. A descrição da teoria *DiffServ* orienta a definição dos requisitos desse tipo de WebLab.

3.1 Teoria de Serviços Diferenciados

Serviços Diferenciados (*Differentiated Services - DiffServ - DS*) é uma abordagem que busca fornecer serviços distintos e escaláveis na Internet sem a necessidade de tratar cada fluxo e nem de sinalizá-los em cada nó [3]. A arquitetura é baseada em um modelo de rede que é implementada sobre um sistema autônomo ou domínio. Com o controle administrativo é possível prover regras consistentes para tratar o tráfego que transita pelo domínio. Esse tráfego pode ser tratado de forma diferente para aplicações de diferentes usuários ou aplicações de usuários de outros domínios.

A arquitetura *DiffServ* é implementada com a classificação do tráfego de entrada no domínio em diferentes classes, também conhecidas como comportamentos agregados (*Behavior Aggregate - BA*) ou agregados. Um agregado é uma coleção de pacotes com a mesma marcação no domínio. Os pacotes que pertencem a uma classe são encaminhados de forma diferente dos pacotes que pertencem a outra. Como exemplo, os fluxos de aplicações de clientes autenticados que precisam de mais recursos de rede podem receber tratamento preferencial sobre os fluxos de outros clientes. De maneira semelhante, os fluxos de lugares desconhecidos podem trazer vírus, *worms*, e-mails de *spam* entre muitos outros, o que provoca um mau uso dos recursos de rede do domínio e prejudica os clientes cadastrados. Esses são alguns dos fatores que justificam a atenção do IETF para o tratamento diferenciado nas redes sem a necessidade de violar o conteúdo dos pacotes.

A classificação de fluxos em classes oferece uma característica básica, mas muito importante dos Serviços Diferenciados: o número de classes tende a ser bem menor do que o número de fluxos. Isso reduz significativamente a quantidade de informação que precisaria ser mantida em cada roteador. São tratadas as classes dos fluxos ao invés dos fluxos individuais, o que reduz a quantidade de recursos para a gerência, otimiza o uso de recursos com uma melhor qualidade do serviço de encaminhamento oferecido e promove maior escalabilidade.

O IETF estudou a possibilidade de diferenciar a distribuição de recursos da rede porque novas aplicações como as de tempo-real que utilizam a Internet como meio de integração de serviços de telefonia fixa, difusão de fluxos de áudio e vídeo, laboratórios virtuais, aplicações de teleconferência e outros mais, exigem um nível maior de qualidade de envio e recepção de informações se comparado

com aplicações convencionais. Ainda que a interligação de redes com TCP/IP permita a entrega de pacotes sem duplicações, perdas e erros, não há garantias quanto à vazão, variação do atraso de envio e recepção (*jitter*), entre outros. Várias propostas de tecnologias foram sugeridas pelo IETF para oferecer QoS na Internet. Dentre elas destacam-se: IntServ/RVSP [55, 56], MPLS (*Multi Protocol Label Switching* [57]) e *Differentiated Services* [58, 59]. *DiffServ* se destacou por promover maior escalabilidade em relação às demais porque realiza o tratamento dos pacotes IP contidos nos fluxos da rede ao invés do tratamento individual desses fluxos. Dentre as necessidades das aplicações de redes destacam-se:

- *vazão (throughput)*: considerando dois *hosts* A e B em uma rede, a *vazão instantânea* em qualquer instante do tempo é a taxa na qual o *host* B está recebendo os dados transmitidos pelo *host* A, geralmente representada em bits/segundo. Caso o *host* B leve T segundos para receber todos os F bits transmitidos pelo *host* A então a *vazão média* da transferência do arquivo é representada por F/T bits/segundo. Portanto, vazão é a taxa na qual o processo que envia dados pode entregar bits ao processo que recebe dados [60].
- *largura de banda (bandwidth)*: largura de banda e vazão são conceitos distintos. Como exemplo, pode-se considerar um cenário onde exista uma conexão entre um *host* servidor e um *host* cliente intermediados por um roteador. Seja T_s a taxa de transmissão do enlace do servidor com o roteador e T_c a taxa de transmissão do enlace do roteador com o cliente. A taxa de envio de dados do servidor não deve ser maior que T_s para evitar perdas de pacotes e atrasos de conexão. De forma semelhante, o roteador não pode encaminhar dados para o cliente a uma taxa maior que T_c . Em virtude disso, a vazão entre o cliente e o servidor será o $\min\{T_s, T_c\}$. Para n outros *hosts* intermediários na conexão entre a origem e o destino, a mesma afirmação é válida, ou seja, a vazão será o $\min\{T_1, T_2, \dots, T_n\}$. A vazão depende das taxas de transmissão dos enlaces que mantêm os fluxos de dados. Os *hosts* da rede e seus elementos de conexão, definem a largura de banda do enlace. Assim, a vazão máxima entre a origem e o destino da conexão pode ser menor ou igual à largura de banda oferecida. Nesse sentido, a largura de banda é a taxa de transmissão máxima oferecida como resultado dos enlaces existentes na comunicação fim-a-fim. Essa taxa é altamente dependente dos elementos intermediários que promovem a conexão [60].
- *perda de pacotes*: quando um pacote encontra a fila do *buffer* do *host* cheia, geralmente ocorre o descarte de pacotes. A fração de perda de pacotes aumenta com a intensidade do tráfego. Isso faz com que a performance de um nó da rede também seja avaliada em função da sua probabilidade de perda de pacotes [60]. Portanto, a perda de pacotes pode ocorrer quando são enviados mais pacotes do que a capacidade de processamento deles nos *hosts* intermediários e/ou no *host* destino, superando a capacidade do *buffer* do *host*. A perda de pacotes dependerá da carga do tráfego, da velocidade relativa do elemento de comutação, da taxa de transmissão do enlace, de erros no encaminhamento dos pacotes, entre outros fatores. Esse é um fator essencial para a análise de congestionamento.
- *atraso fim-a-fim*: é uma medida do atraso total entre a origem e o destino. Seja d_{proc} o atraso de processamento do pacote em cada roteador e no *host* de origem. Seja d_{trans} o atraso de transmissão, cujo valor é L/R , onde L é o tamanho do pacote e R a taxa de transmissão alcançada por cada roteador e pela origem. Finalmente, seja d_{prop} o atraso de propagação em cada enlace. Então, o valor do atraso fim-a-fim é o resultado da fórmula: $d_{fimAfim} = N(d_{proc} + d_{trans} + d_{prop})$, para N-1 roteadores e o *host* de origem. Esse valor é uma aproximação do valor real porque existem diferentes atrasos em cada um dos roteadores, no *host* de origem, e nos enlaces entre a origem e o destino [60].

- atraso de propagação: é o tempo que se leva para propagar um bit de um roteador para o próximo; esse atraso é uma função da distância entre os roteadores, mas nada tem a ver com o tamanho do pacote ou a taxa de transmissão do enlace [60].
- atraso de transmissão: é a quantidade de tempo necessária para o roteador enviar o pacote; esse atraso se dá em função do tamanho do pacote e da taxa de transmissão do enlace, mas nada tem a ver com a distância entre os dois roteadores [60].
- variação do atraso fim-a-fim (*jitter*): em muitas situações de análise de congestionamento e medição de tráfego é interessante conhecer a flutuação do tempo de quando um pacote é gerado na fonte até ele ser recebido no destino. Essa variação do tempo de atraso fim-a-fim é conhecida como *jitter* e irá depender da capacidade de transmissão dos roteadores, do atraso de envio de pacotes do *host* de origem, da largura de banda dos enlaces e também do próprio congestionamento da rede entre a origem e o destino [60].

A implementação de Serviços Diferenciados em uma rede considera que o tráfego de pacotes IP seja classificado e atribuído a diferentes comportamentos agregados. Um agregado é uma coleção de pacotes IP com a mesma marcação no cabeçalho. Podem existir diversos agregados de pacotes sendo encaminhados de acordo com regras específicas através dos nós da rede.

Dentro do domínio DS um Acordo de Nível de Serviço (SLA) define quais recursos de rede estarão disponíveis para os usuários. Esse acordo é definido entre o *host* cliente e o domínio. Também pode ser atribuído um Acordo de Condicionamento de Tráfego (*Traffic Conditioning Agreement - TCA*) que estabelece o que deverá ser feito com os tráfegos de pacotes que não respeitarem o SLA.

3.2 O Campo DS

A arquitetura *DiffServ* é escalável porque utiliza um grupo reduzido de agregados para tratar o encaminhamento de fluxos individuais. Para fazer isso, três quesitos podem ser combinados:

1. marcação: atribuição de bits no cabeçalho do pacote IP;
2. classificação: utilizar esses bits para determinar o tipo de encaminhamento dos pacotes pelos nós da rede;
3. condicionamento: de acordo com as regras de cada serviço preparar os pacotes marcados nas bordas da rede.

O item 1 descreve a atribuição de bits no cabeçalho do pacote IP que é conhecida como marcação. O campo DS (*Differentiated Services field - DS field*) do cabeçalho do pacote IP é o campo que recebe a marcação. Esse é o campo ToS (*Type of Service* para o datagrama IPv4 ou campo *Traffic Class* para IPv6) do cabeçalho do pacote IP [58, 61]. A marcação é feita no campo DS utilizando um código especial conhecido como *Differentiated Services Codepoint* ou DSCP. Esse código é formado por 0s e 1s nos 6 bits mais à esquerda do campo DS. Os bits 6 e 7 não fazem parte da formação do DSCP, mas são utilizados por outras tecnologias para indicar ECN (*Explicit Congestion Notification*). A Fig. 3.1 representa o formato do campo DS.

O item 2 descreve a classificação dos pacotes IP. A marcação e a classificação dos pacotes IP recebe o nome de sinalização. A sinalização é realizada apenas nos roteadores de borda para manter o serviço escalável quanto à necessidade de configuração de recursos nos demais *hosts* do domínio.

Um fluxo de pacotes IP é um conjunto de pacotes IP em trânsito que são identificados pela 5-tupla: endereço origem, porta origem, endereço destino, porta destino e protocolo. Essas informações estão

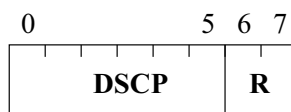


Fig. 3.1: O Formato do Campo DS.

localizadas no cabeçalho do pacote IP. Para selecionar os pacotes IP de um fluxo é necessário utilizar um classificador.

O classificador é um mecanismo que recolhe alguma informação do cabeçalho IP que permite classificar o pacote em alguma classe. Assim, o classificador pode recolher informações como o tipo de protocolo de encaminhamento de pacotes, tipo de serviço (TELNET, SSH, HTTP, entre outros) ou utilizar regras mais complexas por meio da combinação do endereço de origem e destino, por exemplo. Quando são utilizados dois ou mais campos do cabeçalho do pacote IP o classificador é chamado de classificador multi-campo (*Multi-field classifiers*).

O item 3 descreve o condicionamento, ou seja, a preparação dos pacotes IP para que eles respeitem algumas regras. Uma regra pode ser a marcação do cabeçalho de acordo com os outros campos do próprio pacote, o policiamento dos pacotes antes e depois de entrarem no domínio por meio da remarcação ou descarte, entre outros.

3.2.1 PHB

A abordagem *DiffServ* atua no mapeamento do DSCP no cabeçalho do pacote IP [58]. Essa atuação refere-se a um tratamento de encaminhamento particular por nó ou *Per-Hop Behavior* (PHB) ao longo do caminho que o pacote precisa percorrer. Essa definição enfatiza que a atuação *DiffServ* não é realizada fim-a-fim [62]. Os valores para o DSCP podem ser escolhidos de valores recomendados ou terem significado apenas local. Os PHBs são implementados utilizando disciplinas de fila.

Um PHB é um tratamento de encaminhamento que um fluxo de pacotes marcados com um específico DSCP irá receber em cada nó da rede ao longo do domínio [58]. Assim, vários BAs podem ser mapeados para um mesmo PHB.

A priori, apenas o campo DS do pacote IP é usado para determinar o BA ao qual o pacote pertence, mas outros campos também podem ser utilizados para o mesmo fim, com um classificador multi-campo para realizar a seleção. Depois de definir um DSCP para cada BA é necessário realizar o mapeamento deste BA para um PHB. Cada PHB deve estar presente em cada um dos nós da rede do domínio *DiffServ*. Todo pacote que pertence a um BA, mapeado para um PHB, tem de encontrar em qualquer nó o seu PHB implementado para obter o encaminhamento adequado [63]. Essa definição é importante porque implica na preparação de todos os *hosts* do domínio *DiffServ* para o correto encaminhamento diferenciado de serviços.

Cada nó compatível com *DiffServ* deve utilizar os 6 bits do DSCP para o mapeamento de um PHB [58]. Também deve existir um PHB padrão para manter o comportamento de encaminhamento de melhor-esforço (*Best-Effort* - BE). Dessa forma, os fluxos que não se adequem a nenhuma das regras de classificação ainda serão encaminhados utilizando os recursos de rede restantes. O valor recomendado do DSCP para o PHB padrão é '000000'. Os 3 bits mais à esquerda do DSCP são reservados para definir classes de fluxos. Essa restrição garante a criação de até 8 classes de comportamento.

Cada PHB implementado mantém parte dos recursos do domínio. Esses recursos são basicamente o *buffer* e a largura de banda. Um PHB mínimo é considerado aquele que garante uma alocação mínima de largura de banda de determinada porcentagem de um enlace, em um determinado intervalo de

tempo, para um BA. Já um PHB mais complexo deveria garantir uma porcentagem mínima de alocação da largura de banda do enlace com compartilhamento justo proporcional de qualquer capacidade em excesso desse enlace [59].

3.2.2 *Assured Forwarding PHB*

O grupo de PHBs para repasse assegurado (*Assured Forwarding PHB Group - AF PHB Group*) fornece a entrega de pacotes IP em quatro classes AF independentes. Dentro de cada classe AF o pacote IP pode ser atribuído a três níveis diferentes de precedência de descarte, mas mais classes e mais níveis de precedência poderiam ser definidos para uso local [64]. As classes apresentadas aqui não são diferentes das de outrora: as classes anteriores, ou BAs, eram obtidas a partir dos classificadores. As classes AF são os nomes dados às classes anteriores. Isso faz sentido pela definição de que para cada BA deve existir um mapeamento para um PHB.

Dentro de uma classe AF os pacotes IP são marcados com um dos três possíveis valores de precedência (*Drop Precedence - DP*) ou prioridade de descarte. Quando ocorre um congestionamento, a prioridade de descarte de um pacote determina a importância dele na classe: os pacotes de menor prioridade de descarte têm preferência de encaminhamento sobre os de maior prioridade. Ou seja, os pacotes com prioridade de descarte 3 tendem a ser descartados antes dos de prioridade 1, por exemplo.

Essa prioridade é considerada apenas no momento de congestionamento da rede: primeiro são observadas as competições de recursos da classe com as outras; depois são observadas as regras de prioridade para o descarte de pacotes nas sub-classes da classe.

Cada nó deveria implementar todas as quatro classes AF e cada uma delas deve possuir uma quantidade mínima de recursos. Quando mais recursos estiverem disponíveis eles devem ser compartilhados entre as outras classes.

Dentro de uma mesma classe AF o pacote com menor prioridade de descarte terá maior prioridade de encaminhamento em relação aos demais. A representação de um pacote em uma classe AF é feita da seguinte forma: $AFij$ | i =classe, j =precedência de descarte. Os valores recomendados [64] para os DSCP AF são mostrados na Tab. 3.1.

		AF1	AF2	AF3	AF4
Prioridade de Descarte	DP	DSCP / ToS	DSCP / ToS	DSCP / ToS	DSCP / ToS
Baixa	1	001 010 / 0x28	010 010 / 0x48	011 010 / 0x68	100 010 / 0x88
Média	2	001 100 / 0x30	010 100 / 0x50	011 100 / 0x70	100 100 / 0x90
Alta	3	001 110 / 0x38	010 110 / 0x58	011 110 / 0x78	100 110 / 0x98

Tab. 3.1: Prioridade de descarte de pacotes das classes AF.

De acordo com a Tab. 3.1 a classe AF23 possui o DSCP '010110'. Os primeiros 3 bits mais à esquerda representam a classe e, portanto, são os mesmos em todos os DSCP dessa classe. Os 3 bits mais à direita representam a prioridade de descarte ou sub-classe. Nesse caso, o valor binário '110' corresponde à DP 3 e, portanto, o pacote com esse DSCP possui uma alta prioridade de descarte e baixa prioridade de encaminhamento quando ocorrer um congestionamento. Os valores hexadecimais são os valores recomendados pelo IETF para inserir no campo DS do pacote IP.

3.2.3 *Expedited Forwarding PHB*

O PHB de repasse acelerado (*Expedited Forwarding PHB - EF PHB*) pode ser utilizado para disponibilizar uma largura de banda com baixa perda, baixa latência e baixo *jitter* para serviços fim-a-fim através do domínio DS [65]. Um PHB EF pode ser implementado em um roteador de forma

que as regras de encaminhamento assegurem que a taxa de ingresso máximo do BA seja inferior à taxa de egresso mínima desse agregado, ou seja, deseja-se implementar um comportamento de encaminhamento sem filas que, na maioria das vezes, são as responsáveis pelo aumento da perda, latência e atraso da entrega de pacotes.

A taxa de ingresso pode ser gerenciada com o uso de condicionadores de tráfego e, a de egresso, pela própria implementação do EF PHB. O controle de recursos do EF PHB tem prioridade sobre todas as classes AF, mas deve existir um limite de uso desses recursos para não prejudicar os outros fluxos. Assim, o EF PHB garante uma taxa de egresso mínima para algum BA, ou seja, para um conjunto de pacotes IP em trânsito que possuem o mesmo DSCP no domínio DS. O DSCP recomendado é '101 110': os primeiros 3 bits mais à esquerda indicam a classe 5 (complementar às classes AF) e os 3 bits mais à direita representam o número 6, em decimal, para indicar a alta precedência de descarte. O valor hexadecimal recomendado para ser inserido no campo DS é 0xb8.

3.2.4 Arquitetura *DiffServ*

A arquitetura *DiffServ* é baseada em um modelo onde o tráfego de ingresso é classificado, podendo ser acondicionado nos limites da rede, e atribuído a diferentes agregados, ou BAs. Cada BA é identificado por um único DSCP. Os pacotes podem então ser encaminhados de acordo com o PHB associado ao DSCP [59]. Cada nó do domínio deve possuir as mesmas implementações de PHBs, mas a marcação e o condicionamento são realizados apenas nos roteadores de borda da rede. Os nós de borda (ou de fronteira) são os nós de entrada e saída do domínio. Esses nós podem estar conectados a outros domínios *DiffServ* ou não-*DiffServ*. Os nós de núcleo (ou nós interiores) se conectam a outros nós de núcleo e/ou nós de borda. A Fig. 3.2 ilustra a arquitetura *DiffServ*.

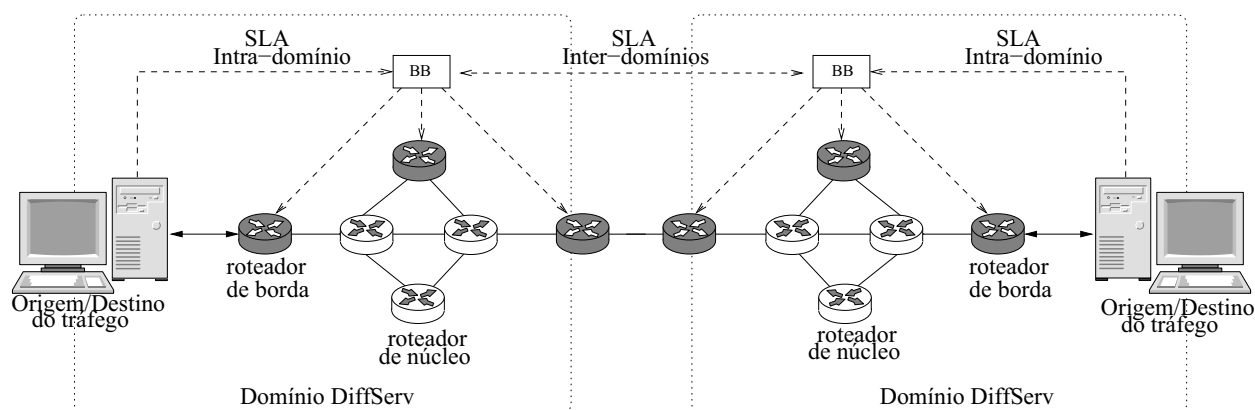


Fig. 3.2: Arquitetura *DiffServ*.

Para promover a escalabilidade no domínio as configurações mais complexas deveriam ser realizadas apenas nos roteadores de borda. Todavia, quando necessário, políticas mais restritivas podem ser realizadas também nos nós de núcleo, mas levando em consideração a preservação da escalabilidade da rede [2]. Os nós de borda atuam como nós *DiffServ* de ingresso e egresso de acordo com as diferentes direções do tráfego. O tráfego entra no domínio através do nó *DiffServ* de ingresso e deixa o domínio através do nó *DiffServ* de egresso. Os nós de borda são responsáveis por assegurar que o tráfego de entrada e saída do domínio respeitem qualquer TCA entre o domínio *DiffServ* e outros domínios. Por outro lado, os Serviços Diferenciados podem ser estendidos para outros domínios *DiffServ* com o estabelecimento de um SLA nos roteadores de borda. O TCA entre os domínios é derivado (explicitamente ou implicitamente) deste SLA [2].

Uma rede compatível com Serviços Diferenciados inclui um classificador que seleciona pacotes de acordo com o valor do campo DS juntamente com mecanismos de gerenciamento de *buffer* e de escalonamento de pacotes [58]. O condicionamento pode ocorrer antes dos pacotes serem marcados (antes dos pacotes entrarem no domínio) ou depois. Assim, não é imposta uma ordem, mas uma combinação de metodologias de medição, marcação, descarte ou atraso da entrega de pacotes. Dentre os processos que podem ser combinados podem ser citados [59]:

- escalonamento (*scheduling*): é o processo pelo qual os pacotes são rearranjados entre a entrada e a saída de um fluxo. As disciplinas de fila são exemplos de implementações do escalonamento;
- marcação (*marking*): é o processo de inserir um DSCP no cabeçalho do pacote IP;
- classificação (*classifying*): é o processo de separação de pacotes em diferentes filas. A classificação pode incluir a marcação.
- medição (*metering*): é o processo de medir propriedades temporais (por exemplo, taxa de transferência) de um fluxo selecionado por um classificador;
- policiamento (*policing*): é o processo de descarte de pacotes de acordo com as informações da medição;
- moldagem de tráfego (*shaping*): o processo de atrasar a entrega de pacotes para respeitar algum perfil de tráfego.
- descarte (*dropping*): o processo de descartar pacotes. Os filtros são capazes de realizar o descarte com base nas informações do policiamento.

Portanto, o condicionamento de tráfego é o processo de alteração do comportamento original do tráfego, seja com medição, policiamento, controle da entrega e/ou marcação de pacotes.

3.3 Visão Geral do *Bandwidth Broker*

O *Bandwidth Broker* (BB) é um agente de redes que gerencia o uso de recursos de rede no domínio *DiffServ*. O BB é capaz de interpretar as requisições de alocação desses recursos e marcar o tráfego de acordo com as políticas estabelecidas [5].

O BB é uma entidade lógica, única por domínio, que realiza a configuração de controle de tráfego nos roteadores, mantém o controle da admissão de tráfego no domínio e faz a negociação automatizada de acordos contratuais entre o cliente e o domínio, e entre diferentes domínios, sejam eles com suporte a *DiffServ* ou não [3].

Para realizar negociações, o BB mantém as informações dos contratos de QoS entre o domínio *DiffServ* e seus clientes. Esse contrato é conhecido como *Service Level Agreement* (SLA) e define a alocação de recursos rede com determinada qualidade de serviço para o cliente em particular [5]. O SLA é um contrato que especifica o serviço de encaminhamento de tráfego que um cliente espera receber. Um *Service Level Specification* (SLS) descreve os recursos fornecidos para um tipo de serviço particular, especificado em um SLA. O SLS contém informações relevantes para o BB e informações sobre os dispositivos de rede de forma a suportar o SLA. Um SLS está associado aos fluxos de dados agregados e à provisão de recursos para esses agregados [66].

O SLA geralmente contém definições informais do tipo de serviço a ser oferecido para o cliente. O SLS é uma tradução formal de como o SLA deve ser aplicado no domínio para atender à QoS solicitada por determinado usuário.

Quando uma alocação é requisitada para um usuário, ela é enviada para o BB. O BB primeiro autentica as credenciais do requisitante para depois verificar se existem recursos não alocados suficientes para atender à requisição. Se a requisição passa por esses testes o BB pode iniciar uma reserva de recursos fim-a-fim ao longo do seu domínio. Os recursos atuais disponíveis são reduzidos pela quantidade requisitada e a alocação da submissão é registrada pelo BB. O BB configura os roteadores de borda da rede de acordo com o perfil de encaminhamento de tráfego.

Atuando como ferramenta administrativa automatizada o BB realiza decisões de controle de admissão de recursos e configuração dos dispositivos de rede [3]. Para gerenciar a qualidade de serviços dentro do domínio *DiffServ* com base no SLA acordado é necessário simplificar a forma como o tráfego é monitorado. O BB auxilia nesse processo ao preparar os roteadores de borda da rede para realizar o tratamento de classes de fluxos, ao invés do tratamento de fluxos individuais.

O BB também monitora os roteadores de borda e os recursos alocados para eles. Com base nessas informações é possível aceitar ou recusar solicitações de alocação de recursos dos clientes do domínio ou de BBs adjacentes. Uma requisição de um cliente para o BB é chamada de *Resource Allocation Request* (RAR). Uma RAR pode conter parâmetros como largura de banda a ser alocada, duração do fluxo e o destino do fluxo. Se a RAR exceder os parâmetros do SLS é função do BB rejeitar a alocação.

Os domínios *DiffServ* podem ser administrados separadamente e ainda assim cooperarem para o fornecimento da alocação dinâmica de QoS fim-a-fim com o uso de múltiplos BBs independentes. O gerenciamento de recursos intra e inter-domínios é uma característica importante presente nas regras de administração do BB.

3.3.1 Gerenciamento de Recursos Intra-domínio

O gerenciamento de recursos intra-domínio controla a alocação e a negociação de recursos dentro de um domínio. Os clientes possuem SLAs que especificam como a alocação de recursos deve ser satisfeita. Cabe ao BB aceitar ou rejeitar uma submissão de alocação de recursos com base na disponibilidade da rede. Dentro do domínio, o BB deve ser capaz de atribuir as configurações *DiffServ* para os roteador de borda de ingresso e egresso da rede.

3.3.2 Gerenciamento de Recursos Inter-domínios

O gerenciamento de recursos inter-domínios controla a alocação e a negociação de recursos nos limites da rede entre dois domínios. O SLA especifica a quantidade e tipos de tráfego a ser enviado/recebido entre os domínios.

É necessário que os pacotes sejam marcados para que os domínios adjacentes sejam capazes de reconhecer quais tipos de tráfego devem receber determinado tipo de controle de tráfego baseado nas políticas estabelecidas no SLA. Portanto, os SLAs precisam ser replicados entre os domínios adjacentes para evitar inconsistências na alocação de recursos entre múltiplos domínios. O BB gerencia os recursos nos enlaces e informa como atribuir os valores no campo DS dos pacotes, antes de encaminhá-los.

3.4 Controle de Tráfego no Linux

As configurações *DiffServ* utilizadas no Linux baseiam-se no controle do tráfego de pacotes IP que percorrem o domínio. Controle de tráfego diz respeito ao conjunto de sistemas de enfileiramento e mecanismos para determinar a taxa com que um *host* pode receber e encaminhar pacotes em uma dada interface [67].

Para realizar o controle de tráfego no domínio *DiffServ* utiliza-se o *software tc* do pacote *IPROUTE2* [68, 67, 69, 70, 71]. Esse *software* interage com o *kernel* do sistema operacional para criar estruturas de controle de tráfego para redes IP. Do ponto de vista conceitual, o *software* encontra-se na Camada de Aplicação do modelo Internet e as configurações atuam na Camada de Rede, uma vez que o tratamento é oferecido para pacotes IP. Assim, os protocolos da Camada de Transporte se aproveitam das configurações realizadas previamente sobre os pacotes que irão trafegar pela rede.

A implementação de um PHB é realizada com o uso de disciplinas de fila (*queue disciplines - qdiscs*). As filas são utilizadas para organizar os pacotes porque os enlaces de rede normalmente encaminham os seus dados em um modelo serial [67]. As *qdiscs* são conhecidas como agendadores (*schedulers*) [63] porque alteram a forma como os pacotes são encaminhados na saída do fluxo.

Uma *classful qdisc* é uma *qdisc* que pode conter classes e fornece mecanismos para associar filtros à disciplina, mas nem todas as *qdiscs* são *classful qdiscs*. As classes são estruturas utilizadas por *classful qdiscs* para encadear outras classes ou *qdiscs*, e estas classes podem estar associadas a filtros. Um filtro implementa a classificação de pacotes e pode ser atribuído a *classful qdiscs* ou classes. As unidades utilizadas pelo comando *tc* são representadas na Tab. 3.2.

<i>Bandwidth e Rates</i>	<i>Quantidade de Dados</i>	<i>Tempo</i>
bps: Bytes/segundo	b: Bytes	s: segundos
kbps: Kilobytes/segundo	kb: Kilobytes	ms: milissegundos (10^{-3} segundos)
mbps: Megabytes/segundo	mb: Megabytes	us: microssegundos (10^{-6} segundos)
kbit: Kilobits/segundo	kbit: Kilobits	
mbit: Megabits/segundo	mbit: Megabits	

Tab. 3.2: Unidades utilizadas pelo comando *tc*.

A ação padrão realizada quando o fluxo de pacotes não respeita os limites de qualquer uma das disciplinas de fila é o descarte de pacotes (*drop*). Por isso é necessário entender o significado dos parâmetros fornecidos na criação dessas disciplinas para evitar perdas de pacotes maiores do que as que seriam obtidas sem o uso do controle de tráfego. A Fig. 3.3 ilustra um exemplo de configuração *DiffServ*. O *script* dessa implementação é apresentado no Apêndice A

3.4.1 *Qdisc* FIFO

A *qdisc* FIFO trata os pacotes igualmente armazenando-os em um única fila e encaminhando-os na mesma ordem em que foram enfileirados. FIFO é a disciplina de fila padrão utilizada pelos sistemas operacionais Linux para promover a entrega de pacotes [67]. A implementação Linux padrão é a *pfifo_fast* que cria 3 bandas (0, 1 e 2) onde são aplicadas, em cada uma delas, regras FIFO para os pacotes. As bandas possuem diferentes prioridades e são capazes de distribuir os pacotes marcados entre elas. Os pacotes da banda 1 apenas são entregues quando não houver mais pacotes na banda 0. O mesmo ocorre com os pacotes da banda 2. Segue um exemplo de uso:

```
# tc qdisc add dev eth1 root bfifo limit 100
```

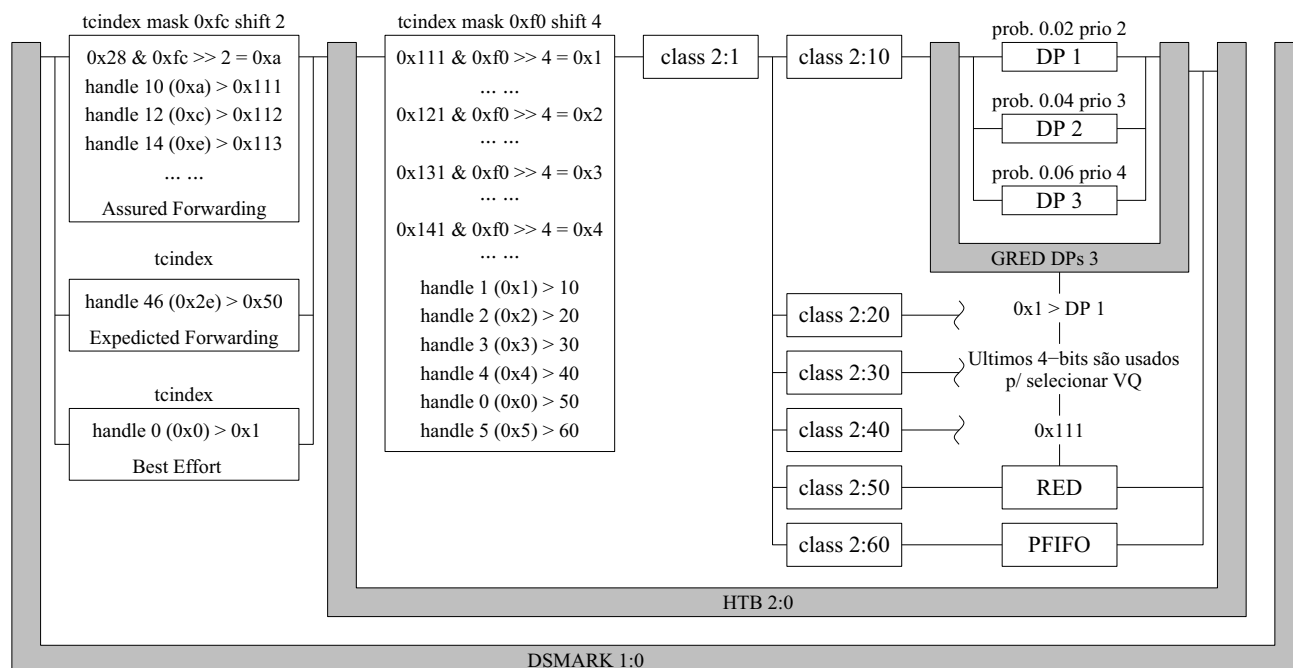


Fig. 3.3: Disciplinas de Fila nos Roteadores de Borda.

Descrição: Adiciona-se a disciplina de fila *bfifo* à interface de rede *eth1*. A fila da *qdisc* é capaz de armazenar até 100 bytes (*limit*). Esse tamanho da fila define a quantidade de informações que podem ser armazenadas caso não seja possível entregá-las na mesma velocidade em que chegam. Cada interface de rede possui uma localização no *kernel* onde podem ser configuradas *qdiscs*. A adição de disciplinas de fila a uma interface é feita a partir da localização raiz (*root*) do dispositivo.

3.4.2 *Qdisc* PRIO

A *qdisc* PRIO é uma *classful qdisc* que permite encaminhar pacotes de diferentes fluxos para diferentes filas de prioridades. Os pacotes classificados em filas de menor prioridade apenas serão entregues quando as filas de maior prioridade não possuírem mais pacotes. Cada uma das filas individuais utiliza o algoritmo FIFO por padrão, o que pode ser um problema: caso apenas uma das filas for preenchida continuamente, as outras não poderão encaminhar os seus pacotes. Por isso, o comportamento padrão das filas pode ser alterado. Segue um exemplo de uso:

```
# tc qdisc add dev eth1 root handle 1:0 prio
# tc filter add dev eth1 parent 1:0 prio 1 protocol ip u32 \
    match ip tos 0x68 0xff flowid 1:3
```

Descrição: Adiciona-se a disciplina de fila PRIO à raiz da interface *eth1* e atribui-se uma numeração (*1:0*) que identifica a *qdisc*. A segunda linha ilustra como o fluxo pode ser atribuído a diferentes classes de prioridade. O parâmetro *parent 1:0* indica que o filtro está em uma hierarquia abaixo da *qdisc* com identificação *1:0*. Essa hierarquia indica que existe dependência entre as configurações. O parâmetro *prio 1* indica a ordem com que os filtros serão buscados para encaminhar os pacotes. Se houvessem outros filtros, a próxima configuração a ser lida pelo comando *tc*, caso não fosse encontrada a correspondência do pacote no primeiro filtro, seria o filtro com *prio 2* e assim por diante. Os parâmetros *protocol ip u32* indicam que será utilizado o protocolo *ip* com o filtro do tipo *u32*. O parâmetro *match ip* indica que será lido o cabeçalho do pacote *ip*. Os parâmetros *tos 0x68 0xff*

indicam que deve ser buscado o valor hexadecimal do campo ToS resultante da operação lógica AND entre os valores 0x68 e 0xff. Transformando os dois valores em binário, 0xff preserva os valores dos bits de 0x68 na operação, como mostrado na Tab. 3.3. Portanto, os pacotes IP que possuem no campo ToS o valor 0x68 serão tratados por esse filtro. Esse é o valor do DSCP para AF31 com dois zeros mais à direita. Finalmente, o parâmetro *flowid* indica que o fluxo será encaminhado para a classe 1:3.

Operação Matemática	Operação Lógica	Interpretação
multiplicação	AND	1 preserva, 0 altera o valor do bit de entrada
soma	OR	1 altera, 0 preserva o valor do bit de entrada

Tab. 3.3: Interpretação das operações lógicas no campo ToS com o comando *tc*.

3.4.3 Qdisc TBF

Pode ser feita uma analogia entre o comportamento virtual da *qdisc* TBF (*Token Bucket Filter*) e o comportamento físico de um balde furado, preenchido com algum líquido: o *buffer* (balde ou *bucket*) é uma fila constantemente preenchida com peças virtuais conhecidas como *tokens* (gotas), em uma taxa de transferência específica (*token rate*). O tamanho do *bucket* é o número de *tokens* que ele pode armazenar. Cada *token* captura um pacote da fila de dados e o remove do *bucket* para realizar o encaminhamento do pacote, semelhante ao furo no balde que deixa vazar o líquido. Um *token* é capaz de manter um conjunto de bytes, o que resolve o problema de armazenamento dos dados de pacotes de tamanhos diferentes. O comportamento do algoritmo TBF permite 3 cenários no *buffer* da interface de rede:

1. Taxa de entrada de pacotes = taxa de recebimento de *tokens*: os dados dos pacotes são atribuídos a *tokens* e não ocorre atraso para a remoção de *tokens* da fila do *bucket*;
2. Taxa de entrada de pacotes < taxa de recebimento de *tokens*: os dados dos pacotes são atribuídos a *tokens*, mas os *tokens* não serão removidos imediatamente do *bucket*. Apenas parte dos *tokens* será removida do *bucket* a cada recebimento de pacotes. Os *tokens* acumulados podem ser usados em rajadas para enviar dados em uma velocidade superior à *token rate*.
3. Taxa de entrada de pacotes > taxa de recebimento de *tokens*: os dados dos pacotes são atribuídos a *tokens*. No entanto, os pacotes são atrasados ou descartados se a fila do *bucket* estiver cheia.

A vazão é obtida com a taxa na qual os pacotes são retirados do *bucket* (parâmetro *rate*). O algoritmo permite o envio de rajadas superiores ao parâmetro *rate* quando os *tokens* ultrapassam o limite do balde. Com a definição de rajadas curtas (parâmetro *minburst*) é possível entregar os pacotes em excesso e evitar o descarte. O parâmetro *peakrate* limita a taxa de envio de rajadas curtas. Segue um exemplo de uso:

```
# tc qdisc add dev eth1 parent 1:3 handle 25: tbf rate 5mbps burst 1mb \
    latency 100ms peakrate 6mbps minburst 1540
```

Descrição: Adiciona-se uma *qdisc* TBF à interface de rede *eth1*. Essa *qdisc* está identificada (*handle*) com a numeração 25: e é filha de uma *qdisc* com identificação 1:3. A vazão atribuída é de 5mbps e um *buffer* de 1mb armazena os *tokens* excedentes. Cada *token* possui um tempo (latência) de 100ms para permanecer no *buffer*. Quando o *buffer* estiver cheio, rajadas podem ser disparadas alcançando a vazão de até 6mbps. A rajada mínima (*minburst*) é de 1540bytes.

3.4.4 *Qdisc* SFQ

A *qdisc* SFQ (*Stochastic Fairness Queuing*) assegura que cada fluxo de pacotes tenha acesso justo aos recursos da rede e previne que rajadas consumam largura de banda além do compartilhamento reservado para o fluxo. O termo “estocástico” refere-se à capacidade do algoritmo de oferecer uma probabilidade de encaminhamento de pacotes com um número limitado de filas para o tráfego. O algoritmo classifica os pacotes em diferentes filas e remove um pacote de cada uma delas por vez, segundo o algoritmo Round-Robin, ou seja, um ciclo que oferece a cada fila uma chance de enviar um pacote.

A SFQ serve os pacotes de cada uma das filas igualmente. Para evitar que os mesmos fluxos sejam servidos sempre nas mesmas filas, o algoritmo que distribui os fluxos é alterado constantemente. Segue um exemplo de uso:

```
# tc qdisc add dev eth1 parent 1:2 handle 20: sfq perturb 10
```

Descrição: Adiciona-se uma *qdisc* SFQ à interface de rede *eth1* e essa *qdisc* é filha da *qdisc* com numeração *1:2*. A *qdisc* SFQ é identificada com a numeração *20:*. De 10 em 10 segundos o algoritmo altera a alocação de fluxos nas filas da disciplina de fila (parâmetro *perturb*).

3.4.5 *Qdisc* RED

A *qdisc* RED (*Random Early Detection*) implementa um algoritmo que informa para a fonte geradora de fluxo uma situação de congestionamento futuro na fila de pacotes IP do destino. Ao invés de descartar apenas os pacotes que chegam quando ocorrer um congestionamento, o descarte é realizado de forma mais igualitária entre todos os fluxos. O objetivo é evitar o congestionamento controlando o tamanho médio das filas e, assim, reduzir o descarte de pacotes. Dessa forma, quando for detectado um estado iminente de congestionamento o tamanho das filas de recepção pode ser alterado para evitar o descarte de muitos pacotes de uma só vez (descarte de rajadas, por exemplo). Segue um exemplo de uso:

```
# tc qdisc add dev eth1 root red limit 64000 min 2600 max 8000 \
    avpkt 1500 burst 3 probability 0.02 bandwidth 300kbps ecn
```

Descrição: Uma *qdisc* RED é adicionada à raiz da interface *eth1*. Para o cálculo dos demais valores são sugeridas [63] as seguintes fórmulas:

```
max = bandwidth em bytes/segundo * latência em segundos
max = 3 * min
limit = 8 * max
burst = (2 * min + max) / (3 * avpkt)
probability = 0.02
```

O parâmetro *limit* informa o tamanho máximo que a fila de bytes pode ter (parâmetro *max* mais o tamanho em bytes de uma rajada) sem descartar pacotes. Os valores de *min* e *max* representam os valores mínimo e máximo, respectivamente, dentro dos quais os tamanhos em bytes das filas podem oscilar. O parâmetro *avpkt* indica o tamanho médio dos pacotes. *burst* é utilizado para permitir uma determinada quantidade de rajadas de tráfego antes de realizar o descarte e o valor de *probability* é a probabilidade máxima de marcação (geralmente 0.02). *bandwidth* é a largura de banda que se quer dispor na *qdisc*. A partir desse valor são calculados a maioria dos outros parâmetros (com exceção do valor de *avpkt* e *probability*). O parâmetro *ecn* é opcional e indica que a fonte receptora deve informar à fonte geradora quanto à uma situação de congestionamento futuro. Os pacotes são marcados ao invés de serem descartados na fonte receptora e pode ocorrer redimensionamento do tamanho da fila RED. A marcação dos pacotes permite à fonte receptora tomar conhecimento da situação.

3.4.6 Qdisc GRED

A *qdisc GRED* (*Generalized Random Early Detection*) foi criada para dar suporte às múltiplas prioridades de descarte requeridas pelo PHB AF. A *qdisc GRED* suporta outras disciplinas de fila em sua hierarquia (*classful*), cada uma delas podendo implementar uma *qdisc RED* com diferentes probabilidades de descarte e com as vantagens inerentes do algoritmo RED. Cada PHB AF precisa implementar uma *qdisc GRED*.

Para utilizar várias *qdisc RED* em uma *qdisc GRED* um campo é adicionado ao *buffer* de pacotes: o campo *tc_index*. Os 4 bits mais à direita desse campo são utilizados para selecionar a fila virtual (*Virtual Queue - VQ*) RED para a qual o pacote pode ser encaminhado.

Antes de atribuir os valores é sugerida [63] a criação de uma tabela como o exemplo da Tab.3.4.

Serviço	VQ	Compartilhamento	Latência	Tamanho médio dos pacotes	Descarte
WWW	1	50%	100ms	512	1%
FTP	2	25%	100ms	1024	1%
VoIP	3	5%	20ms	128	sem descarte
outros	4	20%	100ms	1024	5%

Tab. 3.4: Distribuição esperada de recursos para os pacotes das classes AF.

O cálculo dos parâmetros da *qdisc GRED* é realizado da seguinte forma [63]:

```

limiar máximo =
(0.01 * Largura de Banda Compartilhada *
    Latência *
    Bandwidth em bits) / 8 * 1000
limiar mínimo = 1 / 2 * limiar máximo
avpkt = Tamanho médio do pacote
burst = (2 * limiar mínimo + limiar máximo) / (3 * avpkt)
limit = 4 * limiar máximo
Bandwidth (Ex.: 300Kbits/segundo) =
300 * 1024 bits/segundo = 307.200 bits/segundo

```

A Tab. 3.5 ilustra o resultado da aplicação das fórmulas para um enlace de 300kbps (kilobits por segundo) nos serviços do exemplo. Para os valores fracionados de *burst* foram atribuídos os seus limites superiores. Segue um exemplo de uso:

Serviço	Limiar Máximo	Limiar Mínimo	Burst	Limit
WWW	1920	960	3	7680
FTP	960	480	1	3840
VoIP	38	19	1	153
outros	768	384	1	3072

Tab. 3.5: Resultado da aplicação das fórmulas para a *qdisc GRED*.

```

# tc qdisc add dev eth1 root gred setup DPs 4 default 3 prio
# tc qdisc change dev eth1 root gred limit 7680 min 960 max 1920 \
    burst 3 avpkt 512 bandwidth 300kbit DP 1 probability 0.01 prio 1
# tc qdisc change dev eth1 root gred limit 3840 min 480 max 960 \
    burst 1 avpkt 1024 bandwidth 300kbit DP 2 probability 0.01 prio 2

```

```
# tc qdisc change dev eth1 root gred limit 768 min 96 max 192 \
    burst 1 avpkt 128 bandwidth 300kbit DP 3 probability 1 prio 3
# tc qdisc change dev eth1 root gred limit 3072 min 384 max 768 \
    burst 1 avpkt 1024 bandwidth 300kbit DP 4 probability 0.05 prio 4
```

Descrição: Adiciona-se uma *qdisc* GRED à raiz da interface *eth1* com 4 filas virtuais (*DPs*) com diferentes níveis de prioridade de descarte. A fila virtual padrão é a de *DP 3*. As demais linhas alteram (parâmetro *change*) a *qdisc* GRED uma vez que as filas virtuais fazem parte dessa *qdisc*. Nessas linhas são atribuídos os valores dos parâmetros das tabelas anteriores, calculados com as fórmulas citadas.

3.4.7 *Qdisc* HTB

A *qdisc* HTB (*Hierarchical Token Bucket*) é uma disciplina de fila hierárquica para compartilhamento da largura de banda do enlace. O objetivo é distribuir os recursos do enlace de forma a garantir a largura de banda mínima para uma classe quando ocorrer um congestionamento na rede. Quando forem utilizados menos recursos, a largura de banda restante é distribuída entre as outras classes.

Como ilustrado na Fig. 3.4 o compartilhamento de largura de banda pode ser hierarquizado com vários níveis de distribuição de recursos. No exemplo, o nodo 1:3 ainda pode utilizar até 100% da banda caso o nodo 1:2 não esteja utilizando a rede. A *qdisc* HTB é uma *classful qdisc* e suas classes internas controlam a taxa do fluxo de pacotes que flui por elas. Segue um exemplo de uso:

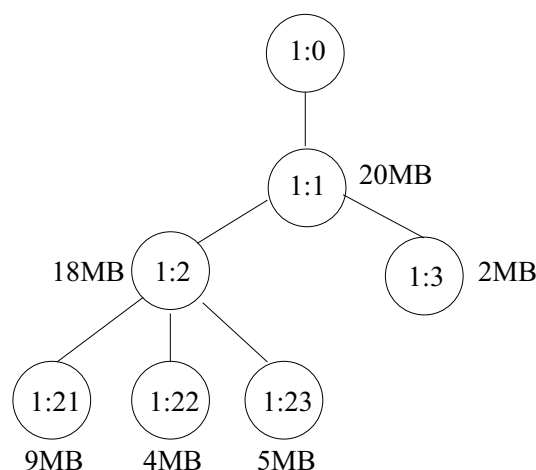


Fig. 3.4: Exemplo de Hierarquia na Configuração da *Qdisc* HTB.

```
# tc qdisc add dev eth1 root handle 1:0 htb
# tc class add dev eth1 parent 1:0 classid 1:1 htb rate 20mbps
# tc class add dev eth1 parent 1:1 classid 1:2 htb rate 16mbps ceil 18mbps
# tc class add dev eth1 parent 1:1 classid 1:3 htb rate 2mbps
# tc class add dev eth1 parent 1:2 classid 1:21 htb rate 9mbps
# tc class add dev eth1 parent 1:2 classid 1:22 htb rate 4mbps
# tc class add dev eth1 parent 1:2 classid 1:23 htb rate 5mbps
```

Descrição: Adiciona-se uma *qdisc* HTB à raiz da interface *eth1*. Seguindo a ilustração da Fig. 3.4 são criadas as classes que distribuem o tráfego. Cada classe referencia a sua classe pai com o parâmetro *parent* e cada classe possui um *classid* único. O parâmetro *rate* define a vazão a ser garantida para classe em momentos de congestionamento. O valor *ceil* é opcional e define o limite superior da vazão. Quando não definido, o valor de *ceil* é o mesmo de *rate*. As classes, no entanto, precisam implementar as disciplinas de fila que irão tratar o encaminhamento do tráfego.

3.4.8 *Qdisc* DSMARK

A *classful qdisc* DSMARK realiza a marcação no campo DS dos pacotes IP. Os pacotes são marcados com valores inteiros e estes são utilizados para definir a classe que irá tratar o pacote. Esses pacotes são marcados apenas antes de eles deixarem a *qdisc* DSMARK. Segue um exemplo de uso:

```
# tc qdisc add dev eth1 root handle 1:0 dsmark indices 2
# tc class change dev eth1 classid 1:1 dsmark mask 0xb8 value 0xb8
# tc class change dev eth1 classid 1:2 dsmark mask 0x1f value 0x60
# tc filter add dev eth1 parent 1:0 protocol ip prio 1 u32 \
    match ip src 172.16.70.1/24 flowid 1:1
# tc filter add dev eth1 parent 1:0 protocol ip prio 1 u32 \
    match ip src 172.16.60.2/24 flowid 1:2
```

Descrição: Adiciona-se uma *qdisc* DSMARK à raiz da interface *eth1* com possibilidade de adicionar até 2 classes DSMARK. As próximas duas linhas utilizam a *qdisc* DSMARK para realizar a marcação. Os pacotes são marcados da seguinte forma:

Campo DS = (Campo DS AND mask) OR value

Seguindo a definição da Tab.3.3, a operação AND utiliza o bit 1 para preservar os bits do campo DS. A operação OR faz o papel inverso para qualquer bit de entrada, ou seja, altera o valor do bit. Com essas duas operações é possível atribuir qualquer valor binário no campo DS. As últimas duas linhas classificam os pacotes e os encaminham para as classes DSMARK com classid 1:1 e 1:2, respectivamente.

Como alternativa para reduzir a quantidade de filtros que explicitamente indicam para qual classe o pacote deve ser encaminhado, foi criado o classificador *tcindex*. Esse classificador realiza operações binárias com uma cópia dos bits no campo DS. A *qdisc* DSMARK faz a cópia desses bits para a estrutura *skb->tc_index* do buffer do pacote IP. Dessa forma, os filtros podem realizar operações binárias utilizando o classificador *tcindex*, que é capaz de ler os bits da cópia. Segue um exemplo de uso:

```
# tc qdisc add dev eth1 handle 1:0 root dsmark indices 2 set_tc_index
# tc filter add dev eth1 parent 1:0 protocol ip prio 1 tcindex \
    mask 0xff shift 2 pass_on
# tc filter add dev eth1 parent 1:0 protocol ip prio 1 \
    handle 0 tcindex classid 1:2
# tc qdisc add dev eth1 parent 1:0 gred setup DPs 2 default 1
# tc qdisc change dev eth1 root gred limit 7680 min 960 max 1920 \
    burst 3 avpkt 512 bandwidth 300kbit DP 1 probability 0.01 prio 1
# tc qdisc change dev eth1 root gred limit 3840 min 480 max 960 \
    burst 1 avpkt 1024 bandwidth 300kbit DP 2 probability 0.01 prio 2
```

Descrição: Adiciona-se uma *qdisc* DSMARK com até 2 classes DSMARK. O parâmetro *set_tc_index* habilita o uso do classificador *tcindex* junto aos filtros. A segunda linha realiza uma operação AND e um deslocamento de bits (divisão por 2). O parâmetro *pass_on* indica que se o primeiro filtro não conseguir tratar o pacote o próximo filtro será utilizado, e assim por diante. *mask*, *shift* e *pass_on* são opcionais. Uma característica da *qdisc* DSMARK é que apenas o menor valor informado para encaminhar o pacote para uma classe específica será copiado para o campo *skb->tc_index*. Por exemplo, para o classid 1:2 apenas o valor 2 será copiado para o campo *skb->tc_index*. A *qdisc* GRED é capaz de ler o conteúdo do campo *skb->tc_index* e encaminhar o pacote para a fila virtual com DP 2.

3.4.9 Policiamento

O policiamento é o processo de descarte de pacotes de acordo com as informações de algum mecanismo de medição do tráfego [59]. O policiamento complementa as configurações *DiffServ* ao evitar que fluxos que não respeitem o perfil de tráfego acordado trafeguem no domínio, descartando os seus pacotes logo nos roteadores de borda da rede, ou realizando a marcação dos pacotes que excederem os limites acordados.

A *qdisc ingress* é uma *qdisc* que habilita o uso de filtros de pacotes. Os filtros são estruturas capazes de realizar o policiamento do tráfego de ingresso e a classificação de pacotes em BAs [67]. A *qdisc ingress* pode utilizar dois tipos de classificadores para agrupar os pacotes IP em BAs: os classificadores *fw* e *u32*.

A ferramenta *iptables* também pode participar do processo de policiamento. Como exemplo, *iptables* pode realizar o processo de marcação de fluxos de pacotes com a mesma origem e, a seguir, os filtros implementam as restrições de ingresso no domínio e a classificação de acordo com a marcação dos pacotes. Segue um exemplo de uso:

```
# iptables -t mangle -A FORWARD -i eth1 -s 172.16.70.0/24 \  
-j MARK --set-mark 3  
# tc qdisc add dev eth1 handle ffff: ingress  
# tc filter add dev eth1 parent ffff: protocol ip prio 1 \  
handle 3 fw police rate 300kbit burst 18k \  
continue flowid 1:1  
# tc filter add dev eth1 parent ffff: protocol ip prio 2 \  
handle 3 fw police rate 100kbit burst 18k \  
drop flowid 1:2
```

Descrição: A ferramenta *iptables* não realiza a marcação do campo ToS do pacote IP, mas a marcação do campo fw do buffer que irá armazenar esse pacote. Portanto, *iptables* realiza a classificação dos pacotes da rede 172.16.70.0/24 e marca o campo fw com o valor 3. A *qdisc ingress* habilita o uso dos filtros. O primeiro filtro utiliza o classificador *fw* para ler o conteúdo do *buffer* do pacote e realiza o encaminhamento na taxa acordada (300kbit) para a classe com *classid 1:1*. O parâmetro *continue* indica que os pacotes que não respeitarem a vazão devem ser tratados pelo próximo filtro capaz de reconhecê-los. Isso é realizado no segundo filtro que trata até 100kbit adicionais do fluxo e os encaminha para a classe com *classid 1:2*. Os pacotes com vazão superior a 400kbit serão descartados (parâmetro *drop*). O parâmetro *prio* indica a ordem na qual os filtros serão buscados pela *qdisc ingress*.

O classificador *u32* também pode ser utilizado para o policiamento. Segue um exemplo de uso:

```
# tc qdisc add dev eth1 handle ffff: ingress  
# tc filter add dev eth1 parent ffff: protocol ip prio 1 u32 \  
match ip tos 0x68 0xff police rate 300kbit burst 18k \  
continue flowid 1:1
```

Descrição: Adiciona-se uma *qdisc ingress* à interface de rede *eth1*. O policiamento ocorre no filtro (*tc filter*) que limita a vazão do fluxo de ingresso a 300kbit por segundo. Os pacotes que possuírem a marcação 0x68 serão encaminhados para a classe com identificação *1:1*.

3.5 Requisitos de um WebLab para Experimentos *DiffServ*

Um WebLab que ofereça suporte para experimentos *DiffServ* deve possuir *hosts* gerenciáveis capazes de interagir com as complexas configurações de controle de tráfego. Essa interação deve ocorrer

apenas nos roteadores de borda da rede: os roteadores de núcleo apenas realizam o encaminhamento do tráfego de pacotes.

A ferramenta *tc* está disponível para sistemas operacionais Linux. Para otimizar o seu uso nos experimentos é útil o uso de *softwares* capazes de simplificar e armazenar as configurações em uma base de dados. Essa solução prioriza o processo de manutenção dos parâmetros fornecidos e reduz o esforço da inserção, remoção e alteração das disciplinas de fila.

O WebLab precisa dispor de pelo menos dois *hosts* comunicantes para experimentos de controle de tráfego, um para a geração e outro para recepção do tráfego. Para experimentos que simulam um domínio *DiffServ*, pelo menos três *hosts* são necessários: dois que desempenham o papel de roteadores de borda e um roteador de núcleo com pelo menos duas interfaces de rede, cada uma estabelecendo um enlace com um roteador de borda adjacente.

O domínio *DiffServ* também precisa de um componente centralizador das configurações que controle o tráfego fornecido ao domínio independente da interação direta com o usuário. Por isso a adição do componente *Bandwidth Broker* [5] ao domínio é de grande valia. Com o uso desse componente, mais ou menos recursos podem ser disponibilizados para os usuários dos experimentos dinamicamente segundo políticas pré-estabelecidas, o que otimiza o uso da largura de banda do laboratório. O *Bandwidth Broker* também realiza a negociação de recursos entre domínios diferentes que podem ser simulados no mesmo WebLab.

Para manter a segurança das alterações o laboratório precisa de uma rede de retaguarda capaz de remover as configurações ao final de cada experimento. O reinício impede que erros no uso do controle de tráfego nos roteadores altere o comportamento normal dos fluxos de outros experimentos. A rede de retaguarda também garante o início consistente da reserva de recursos para os experimentos, quando necessário.

Por outro lado, o uso do aplicativo *tc* exige que o usuário tenha permissões de superusuário para alterar as *qdiscs* das interfaces de rede, mas os aplicativos de interação do usuário com o laboratório devem fornecer acesso restrito às funcionalidades dos recursos.

Para avaliar o resultado das configurações é necessária a análise do fluxo de pacotes com o uso de ferramentas de medição de tráfego. Para isso, o WebLab precisa dispor de recursos que gerem fluxos de pacotes IP. Os experimentos devem ser capazes de reconhecer a origem e o destino dos fluxos e de interromper a transmissão em caso de anomalias. O *log* das informações também deve ser considerado importante para o relato de erros e/ou falhas de execução.

3.6 Considerações Finais

O compartilhamento da largura de banda entre os experimentos é um fator importante e necessário para viabilizar o uso de experimentos simultâneos com diferentes quesitos de QoS. O próximo capítulo descreve as arquiteturas utilizadas nesse trabalho para integrar *DiffServ* a WebLabs de Redes de Computadores.

Capítulo 4

Arquiteturas do NetLab WebLab

O NetLab WebLab é um laboratório remoto de redes de computadores [25]. A característica de ser uma ferramenta de ensino que permite o controle remoto de aplicativos e dispositivos de rede através da Internet inclui esse sistema na categoria de WebLab didático [7]. O NetLab WebLab utiliza o paradigma SOA como solução de integração entre as aplicações remotamente distribuídas.

Dentre os experimentos suportados pela arquitetura do NetLab WebLab destacam-se os experimentos *DiffServ*. Atualmente, a garantia de recursos para aplicações que exigem um fluxo contínuo de informações é crucial e a oferta de serviços diferenciados permite priorizar diferentes tipos de tráfego de dados. Mas essa configuração exige esforço para dispor a rede física que suporte as complexas configurações a serem realizadas. Diante disso, o NetLab foi projetado para permitir a gerência centralizada dos *hosts* com o uso do componente *Bandwidth Broker* e simplificar as configurações para a provisão de recursos fim-a-fim intra-domínio. Os *hosts* que permitem o gerenciamento são configurados para atuarem como roteadores no domínio do laboratório.

O NetLab WebLab suporta diversos experimentos de rede, mas a arquitetura de *software* é genérica o suficiente para que diversos experimentos sejam adicionados com reduzida necessidade de codificação. Essa arquitetura permite a criação de projetos independentes. Como os experimentos utilizam diferentes serviços de interação, a alternativa de separá-los simplifica o processo de manutenção de ambos. A portabilidade da comunicação é oferecida dentro e fora do domínio do laboratório, além de ser mantida a segurança do acesso aos experimentos com o uso dos serviços de acesso.

4.1 Arquitetura SOA do NetLab WebLab

O NetLab WebLab utiliza a arquitetura SOA ao seguir o modelo de referência do projeto GigaBOT [6], utilizando serviços como blocos de construção. Esses serviços são utilizados tanto para a criação do WebLab quanto para a construção de experimentos. Com isso, o NetLab WebLab é capaz de oferecer serviços de gerência do laboratório, de seus participantes e da interação desses participantes com o laboratório.

Essa interação é restringida com o uso de sessões. As sessões limitam o acesso ao WebLab e controlam o uso dos recursos de acordo com o papel atribuído ao participante. Três tipos de sessão são necessários para WebLabs [6]:

- Sessão de Acesso: oferece os mecanismos de controle do acesso ao WebLab e aos experimentos com o uso das credenciais (papéis e permissões) do participante;
- Sessão de Interação: oferece os mecanismos de controle da interação do participante com os recursos remotos (físicos e/ou lógicos);

- Sessão de Comunicação: oferece os mecanismos de controle dos recursos de comunicação. Sistemas de difusão, microfones e câmeras são exemplos desses recursos.

O projeto de um WebLab reúne diversos serviços de gerência. Cada serviço de gerência controla um grupo de serviços que possuem funções relacionadas a uma determinada categoria. Uma arquitetura SOA para WebLabs precisa dos seguintes serviços de gerência [6]:

- Serviços de Gerência de WebLab: gerencia recursos e experimentos;
- Serviços de Gerência de Participantes: gerencia e atribui credenciais aos participantes;
- Serviços de Gerência de Sessões: gerencia as sessões de acesso, interação e comunicação.

Uma sessão utiliza serviços. Para iniciar um experimento, é necessária a criação de uma sessão de acesso, uma ou mais sessões de interação e uma ou mais sessões de comunicação com o WebLab. Para o NetLab WebLab foram efetivamente utilizadas as sessões de acesso e de interação.

4.1.1 Serviços de Acesso

Os serviços de acesso são aqueles que permitem o acesso ao WebLab e ao experimento do WebLab com a autenticação do participante. Nesse processo, as credenciais definem quais serviços podem ser oferecidos para o participante.

Para o uso de experimentos é necessário também a criação de uma sessão de acesso que controla o tempo de uso e verifica periodicamente a disponibilidade de uso do experimento. Por isso são utilizados serviços de acesso para a criação, finalização e verificação do *status* da sessão de acesso.

4.1.2 Serviços de Interação

Os serviços de interação são utilizados pelos experimentos para a interação com os recursos do WebLab. Para o uso de experimentos é necessário também a criação de uma sessão de interação que controla quais recursos devem ser preparados para o uso de um experimento. Por isso são utilizados serviços de interação para a criação e finalização dos recursos alocados para um experimento.

4.2 Gerência Administrativa

Para o controle da gerência administrativa é utilizado o conjunto de aplicativos *Web AccessService* do projeto GigaBOT. Esse projeto centraliza o cadastro de usuários, grupos de usuários, regras de acesso, permissões, laboratórios, experimentos e recursos. Os serviços de acesso desse projeto realizam a autenticação de usuários e o controle de acesso a laboratórios e experimentos.

Para acessar o projeto *AccessService* o usuário utiliza um serviço de acesso para ser autenticado. Um usuário ou grupo de usuários é conhecido como *participante*. Cada participante possui um *papel* que indica como ele pode interagir com o WebLab. Alguns exemplos: aluno, professor, administrador, entre outros. Os papéis possuem diferentes *permissões* para a interação com o laboratório e com os experimentos. Cada participante deve possuir uma identificação, um papel e um conjunto de permissões. Os papéis e as permissões definem as *credenciais* do participante. A Fig. 4.1 ilustra a interface Web do projeto *AccessService*.

O projeto *AccessService* é disponibilizado (*deployed*) no servidor Tomcat como aplicações Web dinâmicas que utilizam JSP (*Java Server Pages*) e *Java Beans*. As páginas JSP interagem com informações armazenadas em uma base de dados MySQL.



Fig. 4.1: Laboratório e seus Experimentos Cadastrados no Projeto *AccessService*.

4.3 Gerência de Uso dos Experimentos

A gerência de uso dos experimentos é realizada com os aplicativos Web do projeto *NetLabWL*, que é uma versão modificada dos aplicativos Web *GigaBOTWL* do projeto GigaBOT. Para acessar um experimento do laboratório o usuário utiliza um serviço de acesso para ser autenticado e, só então, tem acesso à lista de experimentos disponíveis. A Fig. 4.2 ilustra como os experimentos podem ser acessados por meio da interface Web do projeto.

O *NetLabWL* permite visualizar quais experimentos estão disponíveis para o participante autenticado. Ao utilizar a interface Web o participante é capaz de realizar o agendamento de uso do experimento no horário que considerar mais conveniente.

Esse projeto é disponibilizado (*deployed*) no servidor Tomcat como aplicações Web dinâmicas que utilizam JSP e Java *Beans*. As páginas JSP interagem com informações armazenadas em uma base de dados MySQL compartilhada com o projeto *AccessService*.

4.4 Comparação das Arquiteturas de Gerência de Serviços

A Fig. 4.3 ilustra a infraestrutura do WebLab *GigaBOTWL*. Na arquitetura do projeto, os clientes de serviços de interação utilizados pelos participantes fazem parte do projeto *GigaBOTWL*. A exposição das interfaces WSDL dos serviços utiliza *Web Services* disponibilizados no servidor.

No entanto, o desenvolvimento de novos experimentos implica na atualização de um único projeto, o que diminui a disponibilidade de uso do laboratório caso erros sejam encontrados em um serviço de interação particular, impossibilitando o acesso aos demais experimentos. A implementação da funcionalidade de interação é mantida no *Web Service* do respectivo serviço de interação.



Fig. 4.2: Experimentos Disponibilizados para o NetLab WebLab no Projeto *NetLabWL*.

A Fig. 4.4 ilustra parte da arquitetura do NetLab WebLab que mantém a base da arquitetura do projeto GigaBOTWL. O projeto *NetLabWL* separa os experimentos em novos projetos e faz referências a cada um deles. Isso permite que o processo de criação, manutenção e remoção de um experimento seja realizado independente dos demais, sem inviabilizar o uso de outros experimentos no WebLab. Essa arquitetura melhora o processo de distribuição de experimentos entre WebLabs distintos. Cada experimento pode utilizar um ou mais clientes de serviços de interação. A exposição das interfaces WSDL dos serviços é feita com o uso dos *Web Services* disponibilizados no servidor. No entanto, a implementação da funcionalidade da interação com o recurso é mantida apenas no objeto servidor.

4.5 Arquitetura dos Experimentos do NetLab WebLab

Os experimentos do NetLab WebLab são desenvolvidos independentes dos projetos de gerência de serviços. Isso evita que o *site* para o acesso aos experimentos de um WebLab “saia do ar” a cada novo processo de disponibilização e/ou manutenção de experimentos, reduz a dependência entre os projetos e o “espalhamento” de código, e simplifica o processo de distribuição de experimentos entre diferentes WebLabs. Essa solução foi alcançada com a alteração do *software* de gerência de uso dos experimentos do projeto *GigaBOTWL*.

Os clientes dos serviços de interação foram agrupados nos projetos dos experimentos. Como a comunicação utiliza *Web Services*, cada *Web Service* possui um projeto separado dos demais. De forma semelhante, cada experimento é agrupado em um único projeto, separado do projeto de gerência de uso dos experimentos. Os projetos são disponibilizados no servidor separadamente e a criação de novos experimentos é realizada com a composição de *Web Services*: o projeto do experimento apenas informa quais métodos e parâmetros do *Web Service* deseja utilizar. Caso o projeto do *Web Service*

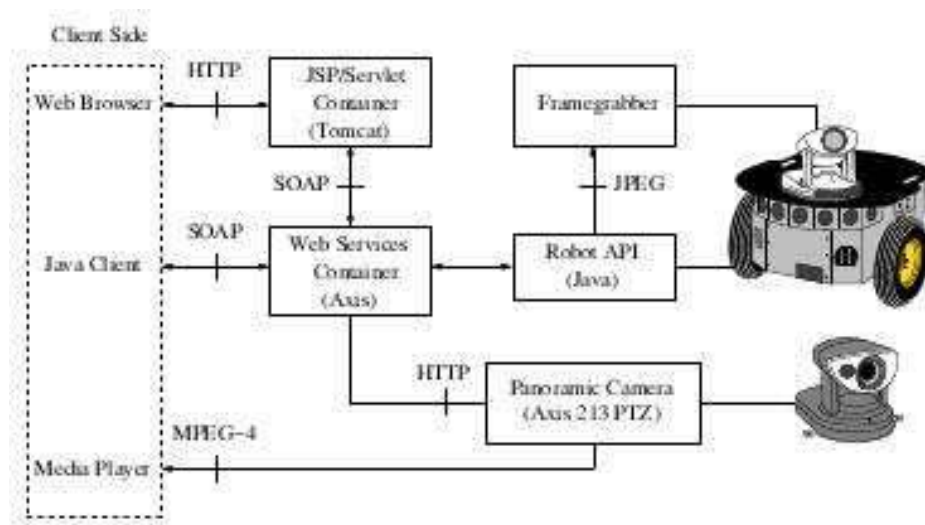


Fig. 4.3: Infraestrutura do WebLab GigaBOT [6].

ainda não tenha sido implementado, o experimento ainda conseguirá ser executado, mas sem executar a ação do *Web Service*.

Um cliente de serviço de interação em um experimento solicita a execução de uma ou mais ações. A comunicação é estabelecida seguindo um modelo composto por três blocos principais, como ilustrado na Fig. 4.5: *bloco cliente*, *bloco de comunicação* e *bloco servidor*. O *bloco cliente* cria o cliente da ação e utiliza uma ou mais interfaces para acessar um ou mais *blocos de comunicação*. O *bloco de comunicação* apenas encaminha para o *host* destino as mensagens que recebe do *host* de origem. Esse bloco desempenha o papel de servidor da requisição cliente e de cliente de uma solicitação para o bloco servidor. Para isso, o *bloco de comunicação* implementa uma ou mais interfaces para acessar diretamente as funções do *bloco servidor* e cada *bloco de comunicação* é específico para um *bloco servidor*. O *bloco servidor* é responsável por processar a requisição e devolver o resultado para o *bloco de comunicação* associado a ele que, por sua vez, encaminha o resultado para o *bloco cliente*.

A arquitetura de um experimento depende de quais funcionalidades estão implementadas com a composição de serviços, como ilustrado na Fig. 4.6. Para iniciar um experimento é necessário que todos os recursos cadastrados sejam previamente inicializados o que se dá com o auxílio de uma fábrica de *software* instanciada em cada *host*. Na inicialização do experimento, o cliente da fábrica comunica-se com o *Web Service FabricaRMI* para que este encaminhe a requisição de criação de instâncias dos objetos que executam a função remota no recurso do laboratório. Depois disso, o cliente do serviço de interação pode interagir com o recurso através do *Web Service* específico associado ao objeto instanciado pela fábrica.

O protocolo SOAP é utilizado quando os componentes da arquitetura encontram-se em domínios diferentes porque esse protocolo simplifica as políticas de segurança para realizar a comunicação entre *firewalls*, *proxies*, *gateways* e demais intermediários de domínios distintos. O protocolo RMI é utilizado para a interação entre os componentes que estão no mesmo domínio para otimizar o desempenho da comunicação na rede interna do laboratório. Mas esse desempenho na comunicação também é otimizado com a redução das funções do *Web Service* que possui apenas métodos para encaminhar as informações entre a origem e o destino.

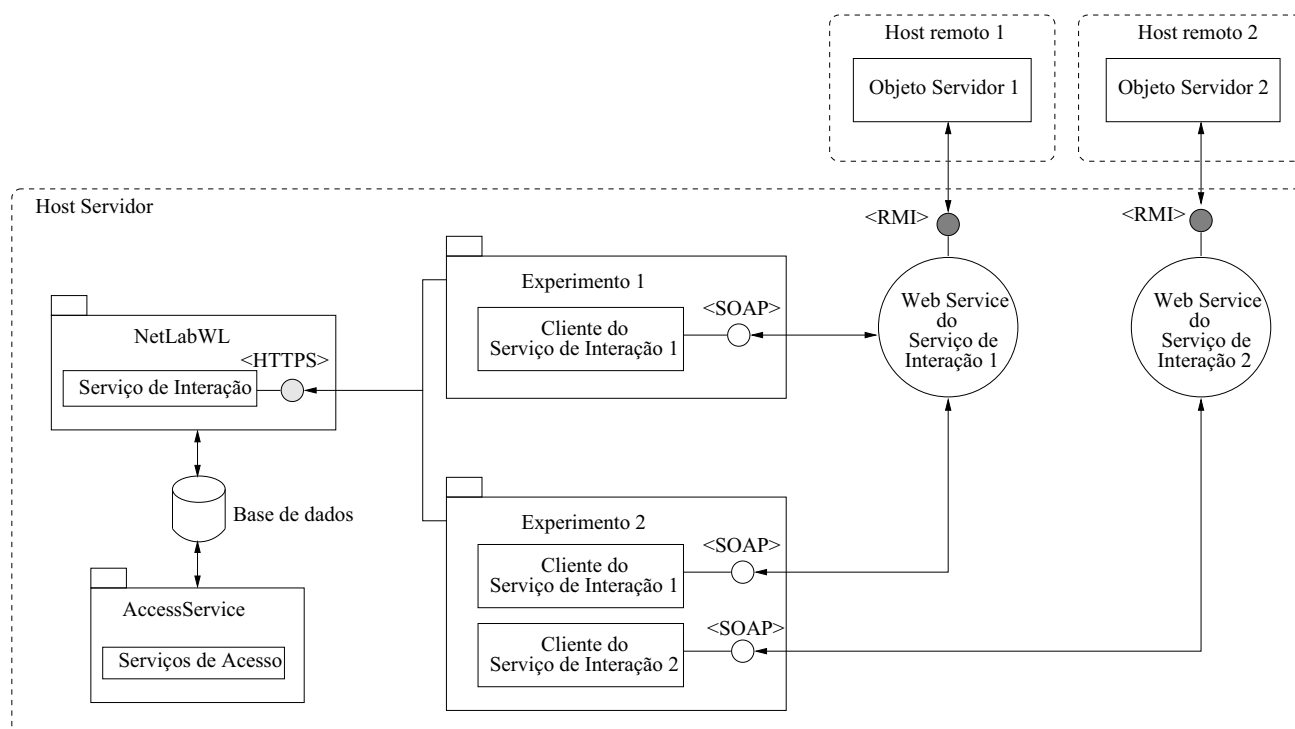


Fig. 4.4: Arquitetura Parcial dos Projetos AccessService e NetLabWL.



Fig. 4.5: Modelo de Blocos para a Realização de Ações em Experimentos.

4.5.1 Composição de serviços em experimentos

A arquitetura da Fig. 4.6 é utilizada em diversos experimentos do NetLab WebLab: experimentos *DiffServ*, configuração de NIC (*Network Interface Card*), tratamento de rotas, teste de conexão entre *hosts*, submissão de fluxos simulados e captura de dados de vídeo. Isso demonstra a praticidade de se seguir a arquitetura para disponibilizar quaisquer tipos de experimentos. Cada *Web Service* está associado a um único objeto servidor que, por sua vez, pode estar associado a um ou mais objetos servidores os quais são responsáveis pela implementação das requisições do aplicativo do experimento e, por isso, podem interagir com outros objetos servidores [72].

A composição de serviços é definida nessa dissertação como um conjunto de serviços utilizados por um experimento, e é realizada como mostra a Fig. 4.7. Para isso, o aplicativo do experimento precisa possuir interfaces para acesso aos outros *Web Services* do laboratório. No entanto, o aplicativo e o *Web Service* do experimento não possuem a lógica para a realização das configurações: essa lógica é implementada apenas pelo objeto servidor instanciado no *host* do laboratório.

O acesso aos aplicativos fora do domínio do WebLab é realizado independente do sistema operacional graças à tecnologia JWS (*Java Web Start*) [73] [74] e não são necessários tratamentos alternativos das informações transmitidas porque o formato das mensagens segue o padrão SOAP. Dentro do domínio, a *performance* dos experimentos é otimizada porque os *Web Services* possuem apenas as referências para os *hosts* do WebLab e para os respectivos métodos nesses *hosts* que executam a ação

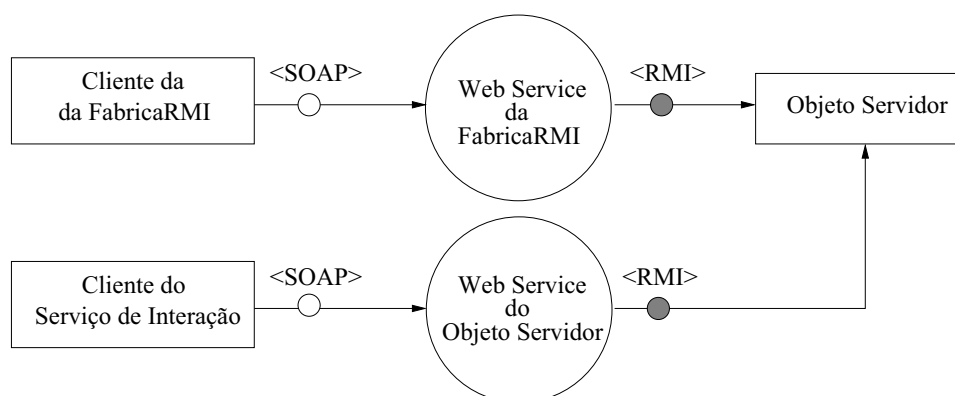
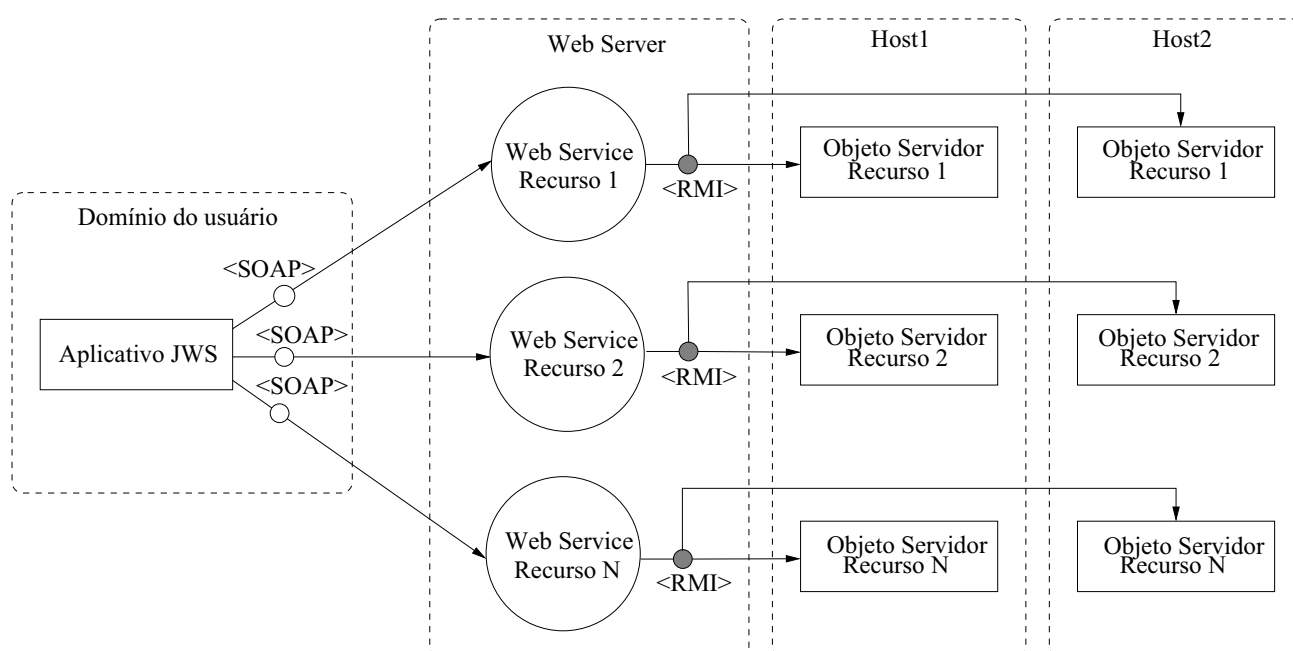
Fig. 4.6: Arquitetura dos Experimentos no *NetLabWL*.

Fig. 4.7: Composição de Serviços no NetLab WebLab.

solicitada pelo usuário. Os *Web Services* fazem então o acesso direto aos métodos dos objetos servidores Java com o uso de RMI. Esses objetos realizam as alterações necessárias nos *hosts*, interagem entre si se necessário e devolvem o resultado para o *Web Service* que, por sua vez, apenas encaminha a resposta para o aplicativo JWS do usuário.

4.6 Serviços de Interação para Experimentos de Redes

A comunicação em cada serviço de interação de um experimento é distribuída em um grupo formado por três blocos: cliente, comunicação e servidor. A implementação de cada um deles é mantida em um projeto separado dos demais.

Os experimentos são aplicações JWS que pertencem ao bloco cliente, os *Web Services* pertencem ao bloco de comunicação e os objetos servidores pertencem ao bloco servidor. Para cada *Web Service* existe um *objeto servidor* específico. A Tab. 4.1 mostra a relação entre os *Web Services* e os experimentos propostos.

Os *Web Services* WSRecursos, WSEnlace e WSSessao são utilizados pelos experimentos de redes

Web Service/Experimento	BB	GeradorFluxos	ServletExp
WSRecursos	X	X	X
WSEnlace	X	X	
WSSessao	X	X	
WSFabricaRMI			X
WSBB	X		
WSGeradorFluxos		X	
WSIfconfig	X	X	

Tab. 4.1: Relacionamento entre Web Services e Experimentos.

disponibilizados para os usuários. O *Web Service* *WSFabricaRMI* é utilizado pela classe *ServletExp* para iniciar os recursos de rede necessários para os experimentos. Os demais *Web Services* são específicos para cada tipo de experimento.

Para que os objetos agrupados nos blocos de interação possam se comunicar eles precisam conhecer as assinaturas dos métodos e a localização remota dos objetos que implementam esses métodos. Essa informação é descrita nas interfaces e *stubs*. Para cada método de interação do experimento com o *Web Service* utiliza-se um código semelhante ao descrito no Apêndice B.1. Ao receber as requisições, o *Web Service* atua como servidor SOAP.

O desenvolvimento de experimentos propôs uma padronização no processo de interação entre os objetos participantes de cada um dos blocos. Quando ocorre uma comunicação com um *Web Service*, o aplicativo cliente precisa conhecer a interface dos métodos remotos. O *Web Service* implementa os métodos da interface e permite o acesso de uma aplicação cliente com o uso de um *stub* (fragmento), que é um representante local do *Web Service* que indica onde e como os seus métodos podem ser acessados remotamente. O processo de desenvolvimento estabelece que tanto o aplicativo cliente quanto o *Web Service* precisam possuir os mesmos *stubs*.

Quando ocorre uma comunicação com um objeto servidor RMI, o aplicativo cliente precisa conhecer a interface dos métodos remotos. O objeto servidor RMI implementa os métodos da interface e permite o acesso de uma aplicação cliente por meio de uma referência remota a eles. O processo de desenvolvimento estabelece que tanto o aplicativo cliente quando o objeto servidor RMI precisam possuir as mesmas interfaces.

O *Web Service* comporta-se como servidor de requisições SOAP do aplicativo JWS e como cliente RMI ao encaminhar as requisições para o objeto servidor RMI. Dessa forma, esse aplicativo precisa possuir um *stub* e uma interface para o objeto servidor. Um exemplo de *Web Service* para adquirir a referência do *host* do laboratório e encaminhar a requisição do cliente é descrito no Apêndice B.2.

A interface deve estar presente no *Web Service* para o acesso aos métodos no objeto servidor RMI e um exemplo é apresentado no Apêndice B.3.

O objeto servidor RMI é a aplicação que atua diretamente no *host* do WebLab e que implementa os métodos da interface. O aplicativo cliente RMI precisa conhecer apenas o nome do *host* onde se encontra o objeto servidor RMI para adquirir uma referência a ele. Quem realiza o intermédio da comunicação e mantém o registro dos objetos servidores RMI instanciados no *host* é o objeto servidor fábrica de objetos. Os objetos servidores definem sua interação externa com a exposição de seus métodos. A comunicação RMI deve utilizar uma interface para permitir o acesso do *Web Service* às funcionalidades dos *objetos servidores*. Uma *interface* é utilizada nas interações RMI para que os *objetos servidores* implementem os métodos declarados.

4.6.1 Serviço de Recursos

O serviço de recursos é um serviço de interação utilizado pelos experimentos de redes para a recuperação de sub-recursos de um recurso físico, como interfaces de rede e câmera associada ao *host*, por exemplo. Para a realização da função desse serviço é utilizado o *Web Service WSRecursos* e o *objeto servidor Recursos*.

WSRecursos

Esse *Web Service* comunica-se com o *objeto servidor Recursos* para recuperar os sub-recursos de um recurso físico. A Fig. 4.8 ilustra o diagrama de classes destacando os três blocos de comunicação (pacotes) para Recursos.

Objeto Servidor Recursos

O *objeto servidor Recursos* é instanciado pela *fábrica de objetos de Retaguarda* que encontra-se no *host* servidor do WebLab. Este objeto provê serviços de acesso à base de dados para recuperação dos sub-recursos de um recurso. Como exemplo, o recurso físico *host* pode possuir diversos sub-recursos que podem ser: NIC, *webcam*, entre outros.

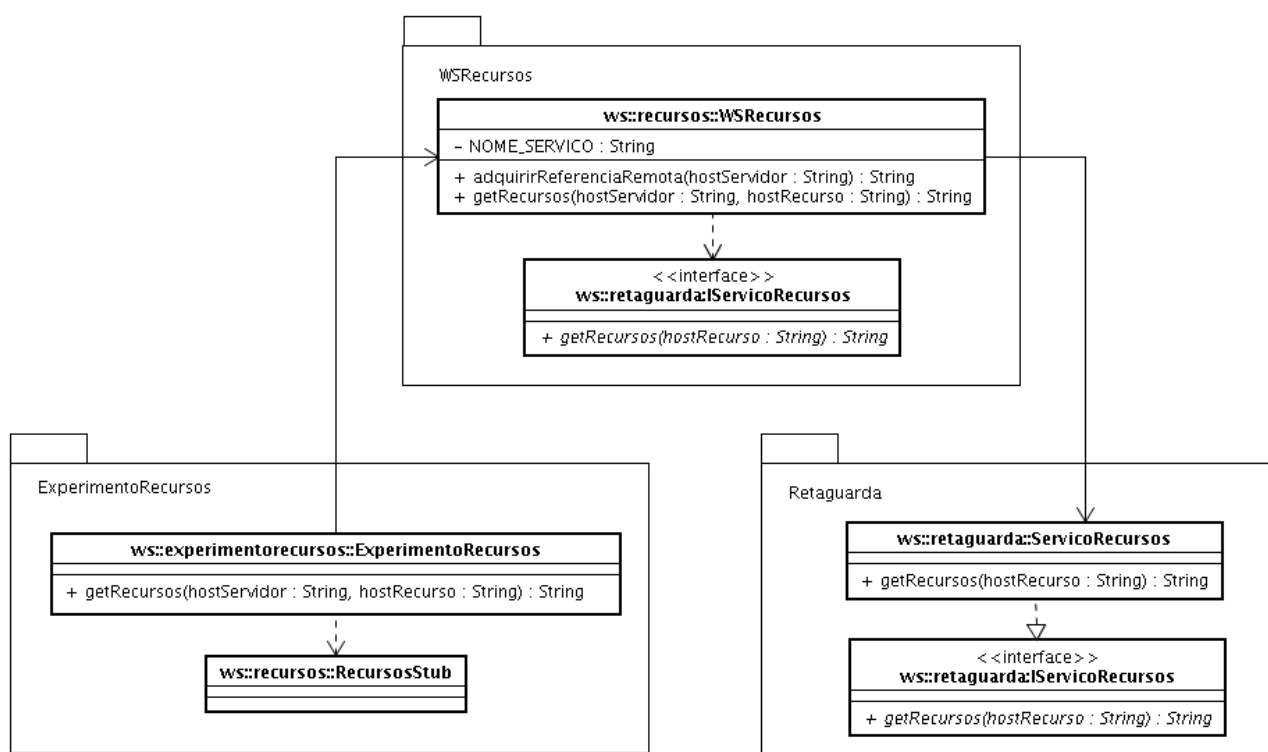


Fig. 4.8: Diagrama de Classes para Recursos.

IServicoRecursos

A interface *IServicoRecursos* declara os métodos expostos pelo *objeto servidor Recursos*. O *Web Service WSRecursos* utiliza essa interface para ter acesso às funcionalidades do objeto servidor. O trecho a seguir ilustra os métodos declarados pela interface.

```

1 package ws.retaguarda;
2 import java.rmi.Remote;
3 import java.rmi.RemoteException;
4 public interface IServicoRecursos extends Remote {
5     public String getRecursos(String host)
6         throws RemoteException;
7 } //fim interface

```

4.6.2 Serviço de Enlace

O serviço de enlace é um serviço de interação utilizado pelos experimentos de redes para a recuperação dos enlaces entre os recursos físicos do experimento. Para a realização da função desse serviço é utilizado o *Web Service WSEnlace* e o *objeto servidor Enlace*.

WSEnlace

Esse *Web Service* comunica-se com o *objeto servidor Enlace* para recuperar os relacionamentos (enlaces) entre os *hosts*. A Fig. 4.9 ilustra o diagrama de classes simplificado para Enlace.

Objeto Servidor Enlace

O *objeto servidor Enlace* é instanciado pela *fábrica de objetos de Retaguarda* que encontra-se no *host* servidor do WebLab. Este objeto provê serviços de acesso à base de dados para recuperação dos enlaces entre os *hosts*. O enlace é estabelecido entre os *hosts* por meio de uma NIC.

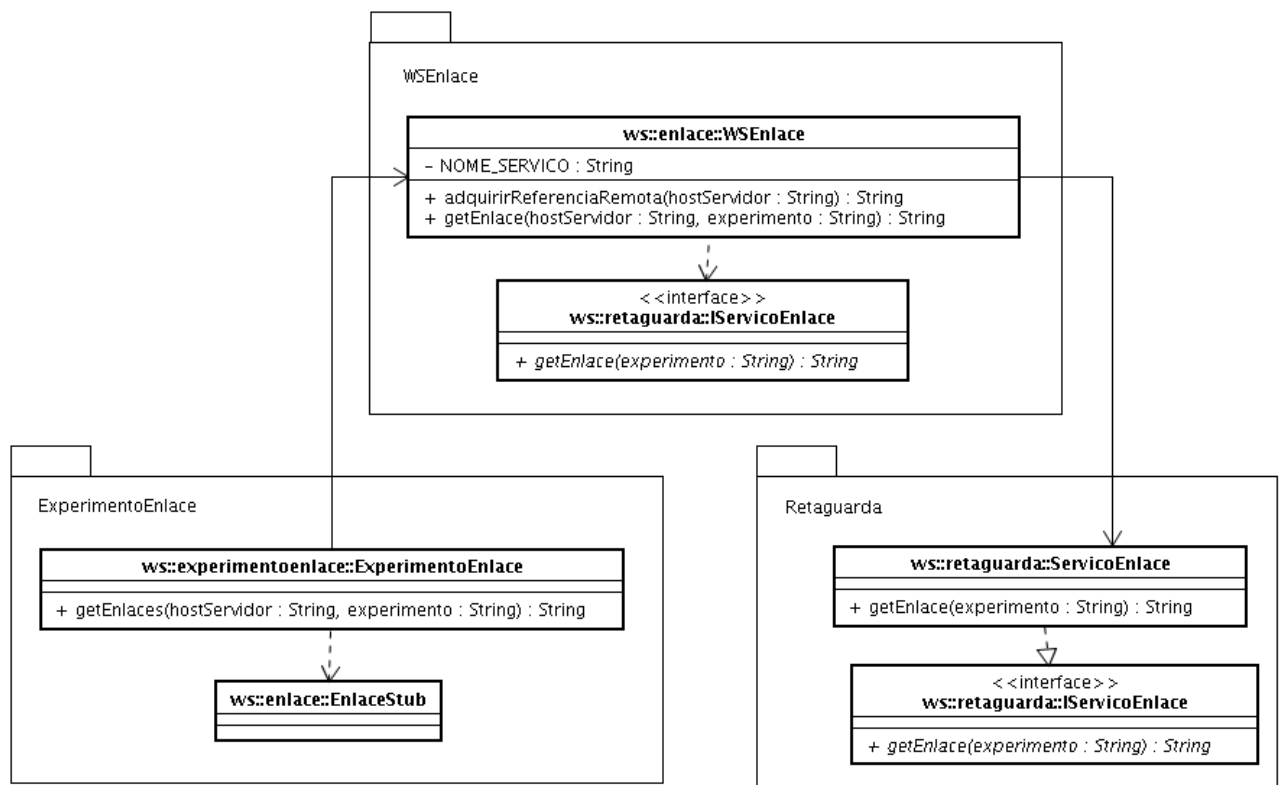


Fig. 4.9: Diagrama de Classes para Enlace.

IServicoEnlace

A interface *IServicoEnlace* declara os métodos expostos pelo *objeto servidor Enlace*. O *Web Service WSEnlace* utiliza essa interface para ter acesso às funcionalidades do objeto servidor. O trecho a seguir ilustra os métodos declarados pela interface.

```
1 package ws.retaguarda;  
2 import java.rmi.Remote;  
3 import java.rmi.RemoteException;  
4 public interface IServicoEnlace extends Remote {  
5     public String getEnlace(String experimento)  
6         throws RemoteException;  
7 }//fim interface
```

4.6.3 Serviço de Sessão de Acesso

O serviço de sessão de acesso é um serviço de acesso utilizado pelos experimentos de redes para o controle do tempo de uso da aplicação e para iniciar, manter e encerrar a sessão de acesso do experimento. Para a realização da função desse serviço é utilizado o *Web Service WSSessao* e o *objeto servidor Sessao*.

WSSessao

Esse *Web Service* comunica-se com o *objeto servidor Sessão* para atribuir e recuperar as informações sobre a sessão de um experimento. O início de uma sessão ocorre a partir do momento que todos os recursos físicos, lógicos e demais configurações tenham sido realizadas com sucesso. Os experimentos devem periodicamente consultar esse *Web Service* para verificar se há tempo suficiente e se o *status* do experimento é válido para a realização de ações no WebLab. A Fig. 4.10 ilustra o diagrama de classes simplificado para Sessão.

Objeto Servidor Sessão

O *objeto servidor Sessão* é instanciado pela *fábrica de objetos de Retaguarda* que encontra-se no *host* servidor do WebLab. Esse objeto retorna o tempo de uso restante de acordo com a reserva do usuário. O *objeto servidor de Sessão* atualiza a cada 2 minutos o *status* de execução de cada experimento iniciado. O experimento iniciado deve, a cada 1 minuto, consultar o seu *status* e gravar a sua atividade na base de dados do *host* servidor do laboratório. Quando o experimento solicita a realização de uma interação com o WebLab é verificado o *status* do experimento. Caso o experimento tenha uma resposta válida, a interação é permitida e o tempo restante da experimentação é retornado. Caso contrário, o experimento é finalizado. Portanto, existem dois estados onde o experimento poderá ser finalizado: um estado ativo e um passivo. Nesses dois casos, a finalização permite a liberação dos recursos do WebLab caso haja inatividade do participante ou se a conexão com esse usuário for interrompida de forma inesperada.

IServicoSessao

A interface *IServicoSessao* declara os métodos expostos pelo *objeto servidor Sessao*. O *Web Service WSSessao* utiliza essa interface para ter acesso às funcionalidades do objeto servidor. O trecho a seguir ilustra os métodos declarados pela interface.

```

1 package ws.retaguarda;
2 import java.rmi.Remote;
3 import java.rmi.RemoteException;
4 public interface IServicoSessao extends Remote {
5     public String gerenciarSessao(
6         String IDExp, String IDUsuario, int opcao)
7         throws RemoteException;
8     public String solicitarUso(String IDBAR)
9         throws RemoteException;
10 } //fim interface

```

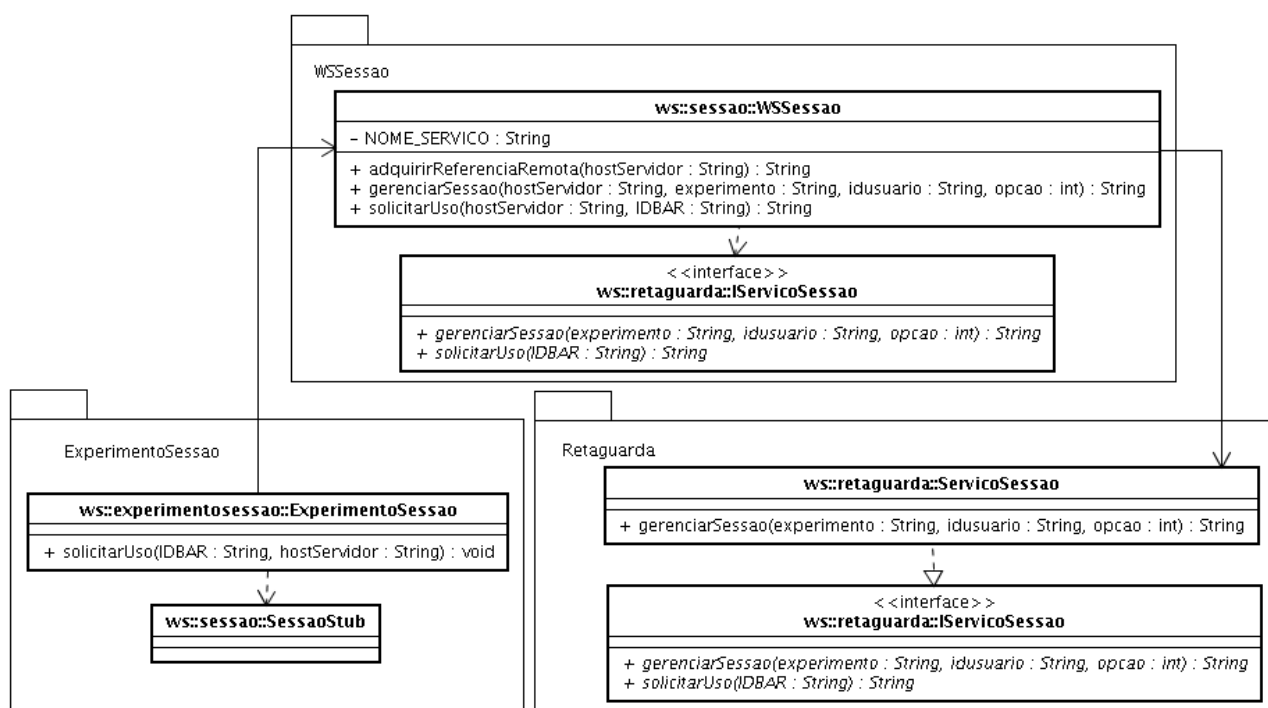


Fig. 4.10: Diagrama de Classes para Sessão.

4.6.4 Serviço de Fábrica de Objetos RMI

O serviço de fábrica de objetos RMI é um serviço de interação utilizado na sessão de interação para a criação e remoção dos objetos servidores alocados para o experimento. Para a realização da função desse serviço é utilizado o *Web Service WSFabricaRMI* e o *objeto servidor FabricaRMI*.

WSFabricaRMI

Esse *Web Service* é utilizado pelo *ServletExp* do WebLab para instanciar todos os recursos físicos e lógicos de um experimento. Esses dados são coletados da base de dados do laboratório. A Fig. 4.13 ilustra o diagrama de classes simplificado para a fábrica de objetos RMI.

Objeto Servidor FabricaRMI

O *objeto servidor FabricaRMI* é uma fábrica de objetos. Uma fábrica de objetos é um objeto capaz de instanciar outros objetos [75]. No processo de interação entre um cliente e um servidor RMI

é necessária a localização do *host* onde se encontram registrados os objetos instanciados remotamente. A fábrica cria esse registro (*rmiregistry*) e se registra nele para permitir que outros objetos possam solicitar serviços da fábrica, como descrito no Apêndice B.4.

A implementação da fábrica de objetos nesse projeto utiliza um único método para iniciar e finalizar os objetos servidores RMI. Quando é feita a solicitação para instanciamento de um objeto pela fábrica, o objeto servidor RMI é instanciado e registrado no *rmiregistry*. Para a finalização, o objeto é removido do registro, como descrito no Apêndice B.5.

No processo de interação entre o cliente e o servidor, o cliente precisa informar apenas a localização do *host* remoto e o método desejado do objeto servidor RMI. No *host* remoto, o objeto da fábrica irá receber a requisição, verificar a existência do objeto servidor RMI no *rmiregistry* e intermediar a comunicação uma vez que o objeto fábrica é o agente responsável por tratar as requisições RMI no *host*. As fábricas de objetos são instanciadas no processo de *boot* de cada um dos *hosts* que atuam como roteadores no laboratório. No servidor é instanciada a *fábrica de objetos de Retaguarda*. Esta fábrica de objetos instancia os objetos que realizam operações no *host* servidor do laboratório. Os objetos servidores de Sessão, Recursos, Enlaces e ColetaDados são exemplos de objetos servidores de Retaguarda. Não é necessário um *Web Service* para essa fábrica porque ela encontra-se no mesmo *host* do WebLab. Nesse caso, a comunicação é realizada por meio de RMI.

IFabricaRMI

A interface *IFabrica* declara os métodos expostos pelo *objeto servidor FabricaRMI*. O *Web Service WSFabrica* utiliza essa interface para ter acesso às funcionalidades do objeto servidor. O trecho a seguir ilustra os métodos declarados pela interface. O método *gerenciarServico* recebe como argumento o nome do objeto servidor. O argumento *opcao* indica se o objeto servidor deve ser ativado ou desativado, com a atribuição dos valores 0 e 1, respectivamente.

```
1 package ws.fabricarmi;
2 import java.rmi.Remote;
3 import java.rmi.RemoteException;
4 public interface IFabricaRMI extends Remote {
5     //Ativa/Desativa o servico no host
6     public String gerenciarServico(String servico, int opcao)
7         throws RemoteException;
8 } //fim interface
```

4.7 Serviços de Interação para Experimentos DiffServ

Os serviços de interação apresentados anteriormente são necessários para realizar experimentos de redes. Para os experimentos *DiffServ* são necessários serviços de interação que realizam funções de controle de tráfego nos roteadores de borda do domínio. Esses experimentos são um subgrupo dos experimentos de redes.

Para os experimentos *DiffServ* foram adicionados um bloco cliente (*Objeto Servidor BB*), um bloco de comunicação (*WSBB*) e um bloco servidor (*Objeto Servidor ServicoTC*). O bloco cliente realiza consultas e operações de persistência. O bloco de comunicação faz o encaminhamento das solicitações entre o *Objeto Servidor BB* e o *Objeto Servidor ServicoTC*. Este implementa as operações de controle de tráfego solicitadas pelo *aplicativo JWS*.

A Fig. 4.11 ilustra a arquitetura que integra os serviços para experimentos de redes com os serviços para experimentos *DiffServ*. Nessa arquitetura são utilizados elementos para recuperar os sub-recursos de um *host* e os enlaces entre eles. Para a implementação da ação de recuperação de enlaces

é necessário o uso do *Objeto Servidor Enlaces* que recupera as informações de enlaces da base de dados. Para realizar a comunicação entre esse objeto e a aplicação cliente deve ser utilizado o bloco de comunicação *WSEnlaces* que encaminha as requisições. De forma similar, esse processo é necessário para a ação de recuperação de sub-recursos.

Para as ações de controle de tráfego são utilizados serviços que informam os parâmetros *DiffServ* com o uso de mensagens SOAP. O *Web Service BB* recebe a requisição e a encaminha para o *Objeto Servidor BB*. Esse objeto é capaz de recuperar da base de dados as informações de quais roteadores de borda devem ser configurados. O objeto envia a resposta para o *Web Service BB* e este a transmite para o *Objeto Servidor TC* em cada roteador de borda do experimento. Esse processo de automatização promovido pelo BB garante a replicação das configurações nos *hosts*. No entanto, cabe ao usuário definir os roteadores de borda e gerar a configuração *DiffServ*.

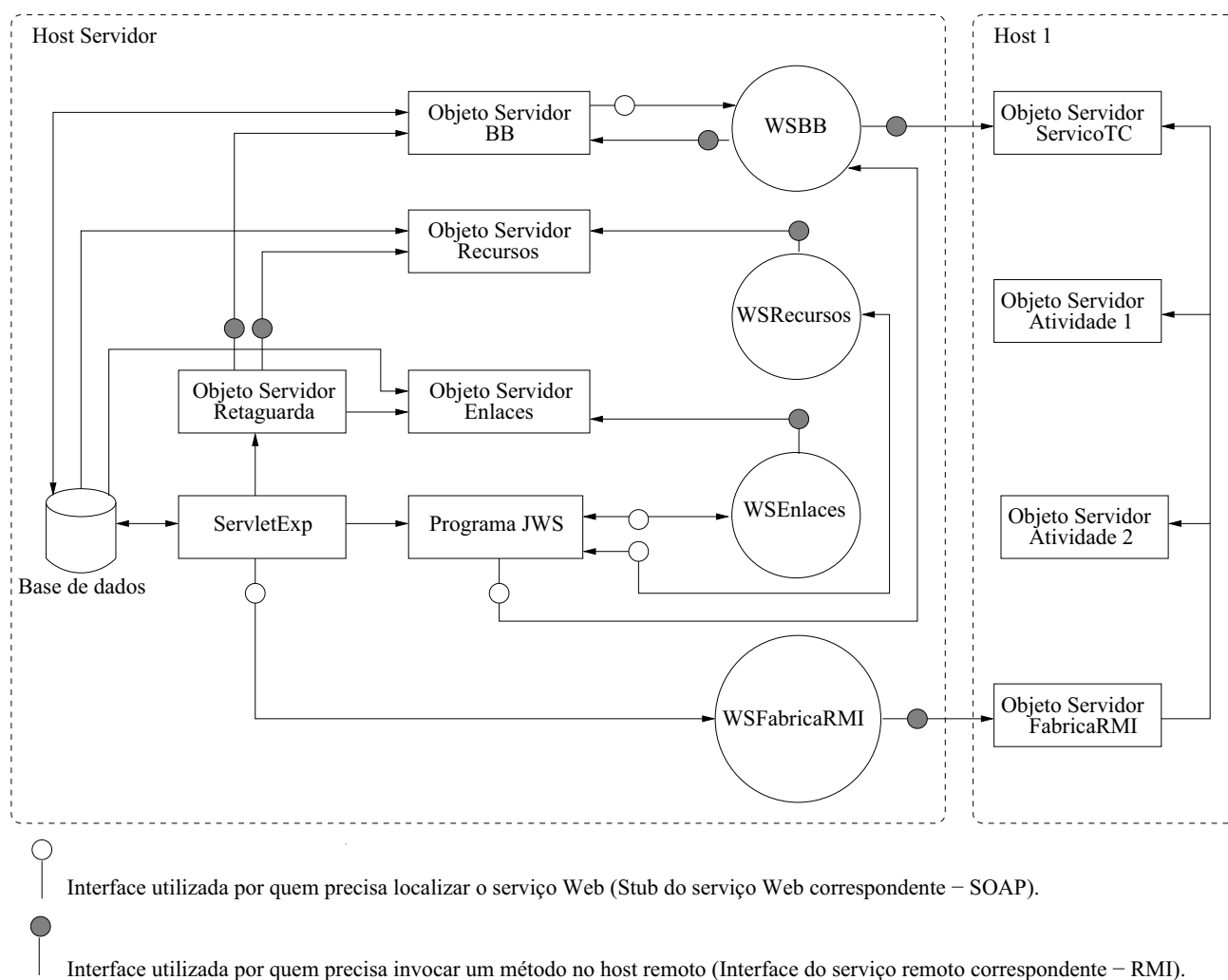


Fig. 4.11: Arquitetura para Experimentos *DiffServ* com Recuperação de Sub-recursos e Enlaces.

Antes de iniciar um experimento o elemento *ServletExp*, presente na aplicação Web de gerência de uso do NetLab WebLab, solicita a instanciação do *Objeto Servidor ServicoTC* em cada um dos *hosts* de borda do experimento. Esse objeto servidor irá realizar as configurações *DiffServ* que forem solicitadas pelo usuário do experimento.

Os *Web Services* comportam-se como clientes RMI quando acessam diretamente os métodos dos objetos servidores RMI nos *hosts* do laboratório e como servidores SOAP quando recebem requisições das aplicações remotas (aplicativo Web de gerência de uso do NetLab WebLab e aplicação

JWS do experimento). Os objetos servidores são instanciados a partir de fábricas de objetos. As fábricas também são objetos servidores porque recebem requisições para a instanciação de outros objetos. Cada objeto é registrado em uma base de dados presente em cada *host*. A fábrica pode criar dinamicamente um ou mais objetos servidores de acordo com a especificidade do experimento.

O experimento BB utiliza o serviço BB que é um serviço de interação necessário para experimentos *DiffServ*. Esse serviço é composto pelo *Web Service WSBB* e pelos objetos servidores *ServicoBB* e *ServicoTC*. O serviço BB atua na inicialização do experimento Gerador de Fluxos para atribuir as configurações *DiffServ* nos roteadores do experimento.

O experimento Gerador de Fluxos utiliza apenas o serviço Gerador de Fluxos para realizar a geração e a coleta de fluxos simulados. Este serviço é formado pelo *Web Service WSGeradorFluxos* e pelos objetos servidores *ServicoRude*, *ServicoCrude* e *ServicoColeta*.

4.7.1 Serviço BB

O serviço BB é um serviço de interação utilizado pelo experimento BB. Esse serviço realiza as funções do agente *Bandwidth Broker* no domínio do laboratório.

Esse serviço de interação utiliza o *objeto servidor ServicoTC* para realizar as configurações de controle de tráfego nos roteadores de borda do experimento, e o *objeto servidor ServicoBB* para realizar as operações de persistência do *Bandwidth Broker* no *host* servidor do WebLab. O *Web Service BB* encaminha as requisições entre esses objetos.

WSBB

Esse *Web Service* é utilizado pelo experimento BB para realizar o gerenciamento das configurações *DiffServ* na rede interna do laboratório. O *objeto servidor TC* realiza as configurações *DiffServ* nos roteadores de borda da rede. O *objeto servidor ServicoBB* é o aplicativo que efetivamente realiza a gerência das configurações no domínio. Este objeto é instanciado pela fábrica de objetos de Retaguarda e realiza operações de persistência do experimento. A Fig. 4.15 ilustra o diagrama de classes destacando os três blocos (pacotes) de comunicação utilizados nesse experimento.

Objeto Servidor ServicoBB

Esse objeto é instanciado no *host* servidor do WebLab. Sua função é implementar a lógica para o gerenciamento da alocação e uso dos recursos *DiffServ* dentro e fora do domínio. O *objeto servidor ServicoBB* é um objeto instanciado pela fábrica de objetos de Retaguarda que realiza operações de persistência no *host* servidor do WebLab. Além de realizar consultas à base de dados, esse elemento da arquitetura do BB armazena as operações de inserção, remoção e atualização das configurações *DiffServ* que são realizadas nos roteadores de borda do experimento.

IServicoBB

A interface *IServicoBB* declara os métodos expostos pelo *objeto servidor ServicoBB*. O *Web Service WSBB* utiliza essa interface para ter acesso às funcionalidades do objeto servidor. O trecho a seguir ilustra parte dos métodos declarados pela interface.

```
1 package ws.retaguarda;
2 import java.rmi.Remote;
3 import java.rmi.RemoteException;
4 public interface IServicoBB extends Remote {
5     public String addQDiscDSMARK_BD(
```

```
6      String IDTC, String ordem, String tipo,
7      String parent, String handle, String indices,
8      String defaultIndex, String setTcIndex
9  ) throws RemoteException;
10 public String delQDiscDSMARK_BD(
11     String IDTC, String ordem, String tipo,
12     String parent, String handle, String indices,
13     String defaultIndex, String setTcIndex
14 ) throws RemoteException;
15 public String queryQDiscDSMARK_BD(
16     String IDTC) throws RemoteException;
17 ...
18 public String addFilterU32_BD(
19     String IDTC, String ordem, String tipo,
20     String parent, String protocol, String prio,
21     String matchIpSrc, String matchIpDst,
22     String matchSport, String matchDport,
23     String matchTos, String rate, String unidRate,
24     String burst, String unidBurst,
25     String acao, String flowid) throws RemoteException;
26 public String delFilterU32_BD(
27     String IDTC, String ordem, String tipo
28 ) throws RemoteException;
29 public String queryFilterU32_BD(
30     String IDTC) throws RemoteException;
31 ...
32 public String addSLA_BD(
33     String IDSLA, String IDUsuario, String IDExp,
34     String QOS, String dataInicio, String horaInicio,
35     String dataFim, String horaFim) throws RemoteException;
36 public String delSLA_BD(String IDSLA) throws RemoteException;
37 public String querySLA_BD() throws RemoteException;
38 public String addPHB_BD(
39     String PHB, String min, String unidMin,
40     String max, String unidMax) throws RemoteException;
41 public String delPHB_BD(String PHB) throws RemoteException;
42 public String queryPHB_BD() throws RemoteException;
43 public String addBandwidthExperimento_BD(
44     String IDExp, String bandwidth,
45     String unidBandwidth) throws RemoteException;
46 public String delBandwidthExperimento_BD(
47     String IDExp) throws RemoteException;
48 public String queryBandwidthExperimento_BD()
49     throws RemoteException;
50 public String addQOSEperimento_BD(
51     String IDExp, String PHB) throws RemoteException;
52 public String delQOSEperimento_BD(
53     String IDExp, String PHB) throws RemoteException;
54 public String queryQOSEperimento_BD() throws RemoteException;
55 public String addExperimentoTC_BD(
56     String IDExp, String host,
57     String dev, String IDTC) throws RemoteException;
58 public String delExperimentoTC_BD(
59     String IDExp, String host, String dev
60 ) throws RemoteException;
61 public String queryExperimentoTC_BD() throws RemoteException;
62 public String addBB_BD(
63     String ip, String bandwidth, String unidBandwidth,
```

```

64     String disponivel, String unidDisponivel
65 ) throws RemoteException;
66 public String delBB_BD(String ip) throws RemoteException;
67 public String queryBB_BD() throws RemoteException;
68 }//fim interface

```

Objeto Servidor ServicoTC

Esse objeto é instanciado nos roteadores de borda do experimento e sua função é realizar a atribuição das configurações de controle de tráfego nesses roteadores. O *objeto servidor ServicoTC* utiliza a ferramenta *tc* para inserir, remover e atualizar as disciplinas de fila, classes e filtros.

IServicoTC

A interface *IServicoTC* declara os métodos expostos pelo *objeto servidor ServicoTC*. O *Web Service WSBB* utiliza essa interface para ter acesso às funcionalidades do objeto servidor. O trecho a seguir ilustra os métodos declarados pela interface.

```

1     package ws.fabricarmi;
2     import java.rmi.Remote;
3     import java.rmi.RemoteException;
4     public interface IServicoTC extends Remote {
5         public String addQDiscDSMARK(
6             String dev, String parent, String handle,
7             String indices, String defaultIndex,
8             String setTcIndex) throws RemoteException;
9         public String delQDiscDSMARK(
10            String dev, String parent, String handle,
11            String indices, String defaultIndex,
12            String setTcIndex) throws RemoteException;
13        ...
14        public String addFilterU32(
15            String dev, String parent, String protocol,
16            String prio, String matchIpSrc,
17            String matchIpDst, String matchSport,
18            String matchDport, String matchTos,
19            String rate, String unidRate, String burst,
20            String unidBurst, String acao,
21            String flowid) throws RemoteException;
22        public String delFilterU32(
23            String dev, String parent, String protocol,
24            String prio, String matchIpSrc,
25            String matchIpDst, String matchSport,
26            String matchDport, String matchTos,
27            String rate, String unidRate, String burst,
28            String unidBurst, String acao,
29            String flowid) throws RemoteException;
30    }//fim interface

```

4.7.2 Serviço de Geração de Fluxos

O serviço de geração de fluxos é um serviço de interação utilizado pelo experimento Gerador de Fluxos. Esse serviço recebe o tempo de submissão, a qualidade do serviço do fluxo e a vazão que se deseja submeter à rede.

Para a realização da função desse serviço é utilizado o *Web Service WSGeradorFluxos* e os *objetos servidores*: *ServicoRude*, *ServicoCrude* e *ServicoColeta*.

WSGeradorFluxos

Esse *Web Service* é utilizado pelo experimento Gerador de Fluxos para promover a comunicação com os objetos servidores *ServicoRude* e *ServicoCrude*. A Fig. 4.14 ilustra o diagrama de classes destacando os três blocos (pacotes) de comunicação utilizados nesse experimento.

O *pacote GeradorFluxos* reúne um conjunto de classes *stub* que são utilizadas pela classe principal do projeto do experimento para se comunicar com os *Web Services*. O *pacote WSGeradorFluxos* recebe e encaminha as requisições da aplicação cliente, mas precisa das interfaces dos objetos servidores *ServicoRude* e *ServicoCrude* para proceder com a requisição. As interfaces permitem ao *Web Service* ter acesso às funcionalidades dos objetos servidores instanciados pela fábrica de objetos em cada um dos *hosts* que participam do experimento. Finalmente, o *objeto servidor ServicoColeta* do *pacote de Retaguarda* é acessado pela classe principal do experimento para realizar a coleta de dados no *host* servidor do WebLab.

Objeto Servidor ServicoRude

Esse objeto é responsável por encaminhar fluxos para um *host* destino na rede interna do WebLab. Os argumentos recebidos por esse objeto são transformados em um script para o aplicativo RUDE [76].

IServicoRude

A interface *IServicoRude* declara os métodos expostos pelo *objeto servidor ServicoRude*. O *Web Service WSGeradorFluxos* utiliza essa interface para ter acesso às funcionalidades do objeto servidor. O trecho a seguir ilustra os métodos declarados pela interface.

```
1 package ws.fabricarmi;
2 import java.rmi.Remote;
3 import java.rmi.RemoteException;
4 public interface IServicoRude extends Remote {
5     public String receberFluxos(
6         String hostDestino,
7         String IPDestino,
8         String tipo,
9         String fluxos) throws RemoteException;
10 } //fim interface
```

Objeto Servidor ServicoCrude

Esse objeto é responsável por coletar os fluxos gerador pelo *objeto servidor ServicoRude* em um *host* da rede interna do WebLab. O *objeto servidor ServicoCrude* utiliza o aplicativo CRUDE [76] para realizar a coleta dos fluxos. Esse objeto também é responsável por ativar a plotagem dos valores recolhidos. A plotagem é realizada com o uso do *software* Qosplot [77].

IServicoCrude

A interface *IServicoCrude* declara os métodos expostos pelo *objeto servidor ServicoCrude*. O *Web Service WSGeradorFluxos* utiliza essa interface para ter acesso às funcionalidades do objeto servidor. O trecho a seguir ilustra os métodos declarados pela interface.

```
1 package ws.fabricarmi;  
2 import java.rmi.Remote;  
3 import java.rmi.RemoteException;  
4 public interface IServicoCrude extends Remote  
5     public String ativarCrude() throws RemoteException;  
6     public String desativarCrude() throws RemoteException;  
7     public String ativarPlotagem() throws RemoteException;  
8     public String recolherLogCrude() throws RemoteException;  
9 //fim interface
```

Objeto Servidor ServicoColeta

Esse objeto é utilizado pelo experimento Gerador de Fluxos para realizar a consulta periódica do fim da execução da geração de fluxos, coleta dos resultados da submissão de fluxos e fim da plotagem. Finalmente, esse serviço também é responsável pela transferência das plotagens realizadas no *host* destino para o *host* servidor do WebLab. Dessa forma, o aplicativo Gerador de Fluxos pode realizar as consultas sobre as plotagens em um *host* fixo do WebLab.

IServicoColeta

A interface *IServicoColeta* declara os métodos expostos pelo *objeto servidor ServicoColeta*. O *Web Service WSGeradorFluxos* utiliza essa interface para ter acesso às funcionalidades do objeto servidor. O trecho a seguir ilustra os métodos declarados pela interface.

```
1 package ws.retaguarda;  
2 import java.rmi.Remote;  
3 import java.rmi.RemoteException;  
4 public interface IServicoColeta extends Remote  
5     public String consultarFimRude() throws RemoteException;  
6     public String consultarFimLogCrude() throws RemoteException;  
7     public String consultarFimPlotagem() throws RemoteException;  
8     public String recolherPlotagem() throws RemoteException;  
9 //fim interface
```

4.8 Processos de Início e Término de Experimentos

O processo de inicialização garante que todos os recursos necessários para o correto funcionamento do experimento serão ativados e iniciados em um estado consistente. Isso ocorre antes do aplicativo para a interação com o laboratório ser completamente *downloaded* no *host* do usuário. O processo de finalização realiza o papel inverso, ou seja, reinicia o estado dos recursos com os valores de antes da interação e finaliza os elementos instanciados nos *hosts* do laboratório. A Fig. 4.12 ilustra a arquitetura para os processos de início e término de um experimento. Essa arquitetura permite adicionar novos elementos à sua estrutura sem causar interferência nos experimentos que já utilizam os elementos existentes. A comunicação entre os elementos de domínios distintos respeita o modelo de blocos da Fig. 4.5. São utilizadas duas fábricas de objetos: uma para instanciar objetos servidores que recebem requisições para interação com os *hosts* do laboratório e outra para instanciar objetos servidores que recebem requisições para iniciar os objetos servidores criados pela fábrica anterior. A primeira fábrica recebe o nome de *FabricaRMI* e a segunda de *Retaguarda*.

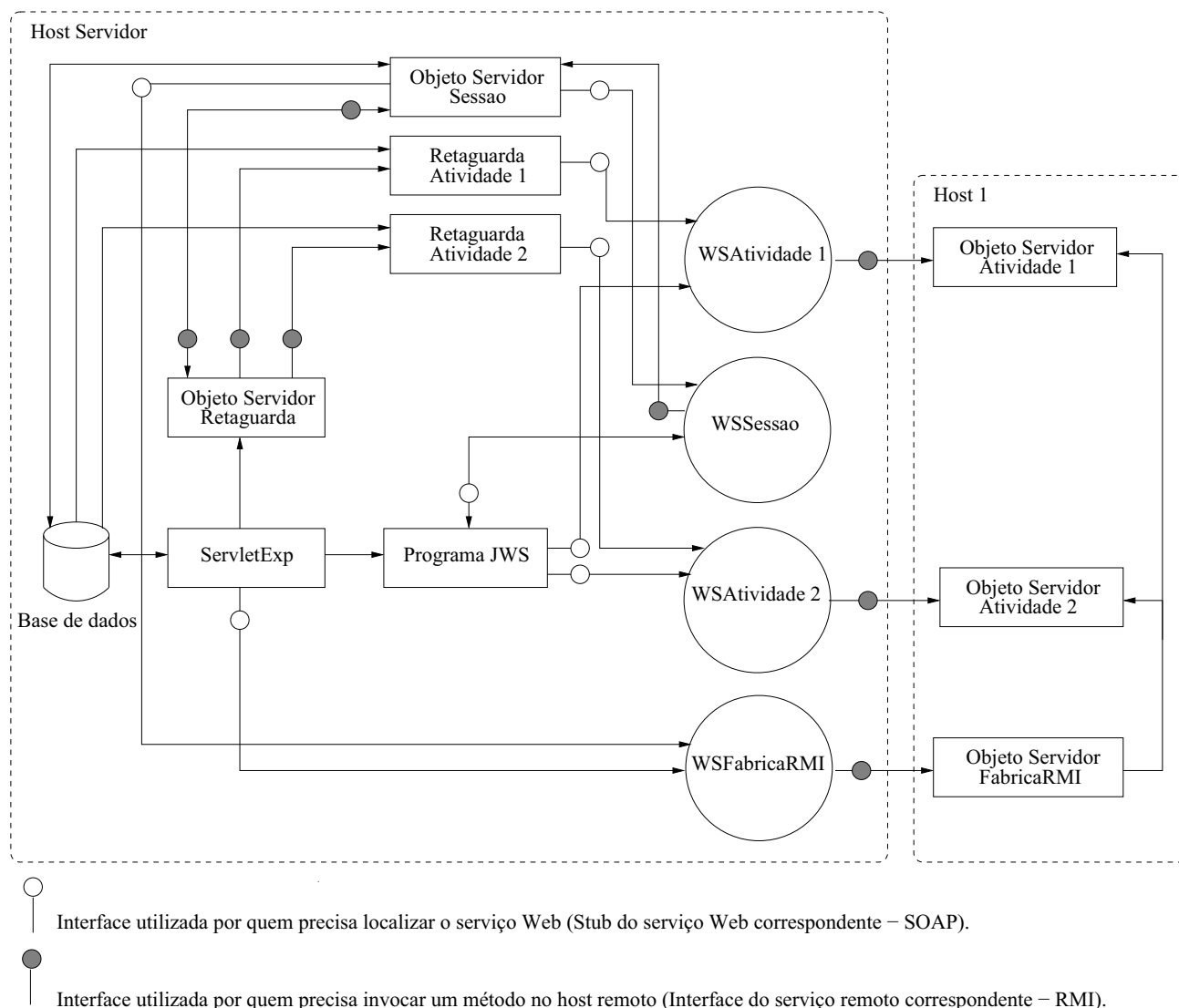


Fig. 4.12: Arquitetura para Início e Término de Experimentos no NetLab WebLab.

4.8.1 Início de Experimentos

O primeiro elemento a ser observado é o *ServletExp*. Quando o participante deseja acessar o experimento ele deve fornecer o seu nome de usuário e senha para que um serviço de acesso do *site* do laboratório realize a autenticação do usuário. Caso a autenticação seja válida, o elemento *ServletExp* realiza o processo de verificação da disponibilidade de uso do experimento pelo usuário. A reserva é realizada pelo próprio usuário com um serviço de interação que realiza o agendamento. O elemento *ServletExp* continua o seu processo realizando uma consulta na base de dados pelos recursos do experimento. Os recursos são divididos em dois tipos: físicos e lógicos. Os recursos físicos são os *hosts* e os lógicos são os objetos servidores que recebem requisições de interação com os recursos físicos e retornam o resultado para o cliente.

Após essa consulta, o *ServletExp* comunica-se com o elemento *WSFabricaRMI* que é o *Web Service* que interage com o objeto servidor *FabricaRMI* em cada um dos *hosts* do experimento. Nessa interação são instanciados todos os recursos lógicos (objetos servidores de atividades) em cada um dos *hosts* do experimento. Em cada *host* são instanciados os mesmos objetos servidores de atividades. Então o *ServletExp* armazena na base de dados as informações sobre o início da experimentação.

O processo continua com a inicialização dos *hosts* em um estado consistente. Isso é feito com

o *Objeto Servidor Retaguarda* que instancia um *objeto de retaguarda* para cada *objeto servidor de atividades* instanciado nos *hosts* do laboratório. Cada *objeto de retaguarda* interage com um *Web Service* específico que, por sua vez, atribui os valores iniciais ao seu *objeto servidor de atividades*.

O *Objeto Servidor de Retaguarda* também instancia o elemento *Sessão* responsável pelo controle do tempo de uso do experimento. Quando o *aplicativo JWS* é iniciado ele solicita, periodicamente, as informações sobre o tempo restante da experimentação por meio do *objeto Sessão*. Esse envio é realizado utilizando o *Web Service WSSessao*. Após a preparação do ambiente, o *aplicativo JWS* pode ser *downloaded* no *host* do participante.

4.8.2 Finalização de Experimentos

O processo de finalização ocorre em uma das seguintes situações:

- término da *aplicação JWS* pelo usuário: quando o usuário finaliza remotamente a sua aplicação;
- término da aplicação por meio do *objeto servidor de Sessão* no *host* servidor do laboratório: o término de um experimento também é controlado pelo laboratório com o uso de aplicações que interagem com o *objeto servidor de Sessão*;
- término do tempo reservado para uso do experimento: o *objeto servidor de Sessão* dispara o evento de finalização do experimento;
- inatividade ou exceção da experimentação: o *objeto servidor de Sessão* atualiza periodicamente informações sobre o tempo restante para uso de cada experimento ativo. Nesse envio, caso ocorra um *timeout* da confirmação da recepção, o *objeto servidor de Sessão* dispara o processo de finalização do experimento.

O processo de finalização solicitado pela *aplicação JWS* envia uma mensagem de término para o *Web Service WSSessao* que encaminha a requisição para o *Objeto Servidor de Sessão* no *host* servidor do laboratório. As demais situações são iniciadas nesse objeto. A primeira tarefa dele é realizar a operação de persistência ao armazenar na base de dados as informações de término.

Logo após o *Objeto Servidor de Retaguarda* é contactado para solicitar a cada um dos *objetos de retaguarda* a reinicialização dos valores atribuídos aos *hosts*, com base nas informações armazenadas na base de dados. Cada um desses objetos sabe como reinicializar seus respectivos objetos remotos. Feito isso, o serviço de retaguarda finaliza cada um dos objetos instanciados no *host* servidor.

O próximo passo é solicitar a finalização dos *objetos servidores* dos *hosts*. Para isso, o *Objeto Servidor de Sessão* precisa de uma interface para informar ao *WSFabricaRMI* os *objetos servidores de atividades* que precisam ser finalizados nos *hosts* do laboratório.

4.9 Considerações Finais

A arquitetura do NetLab WebLab orienta a implementação de experimentos modulares com suporte a *DiffServ* utilizando a base da infraestrutura do projeto GigaBOT. O próximo capítulo descreve a implementação dessa arquitetura e a validação da proposta intra-domínio.

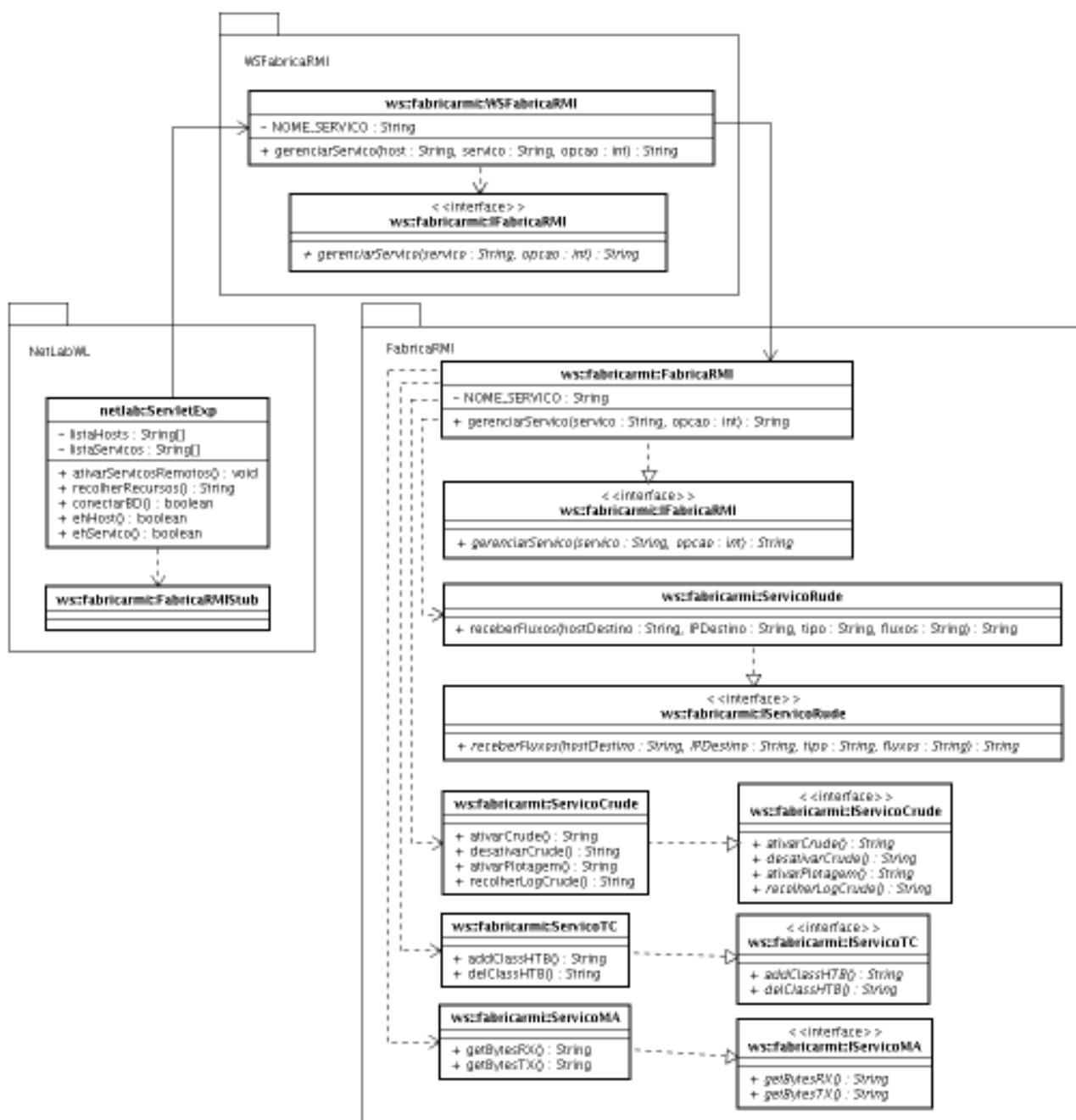


Fig. 4.13: Diagrama de Classes para a Fábrica de Objetos RMI.

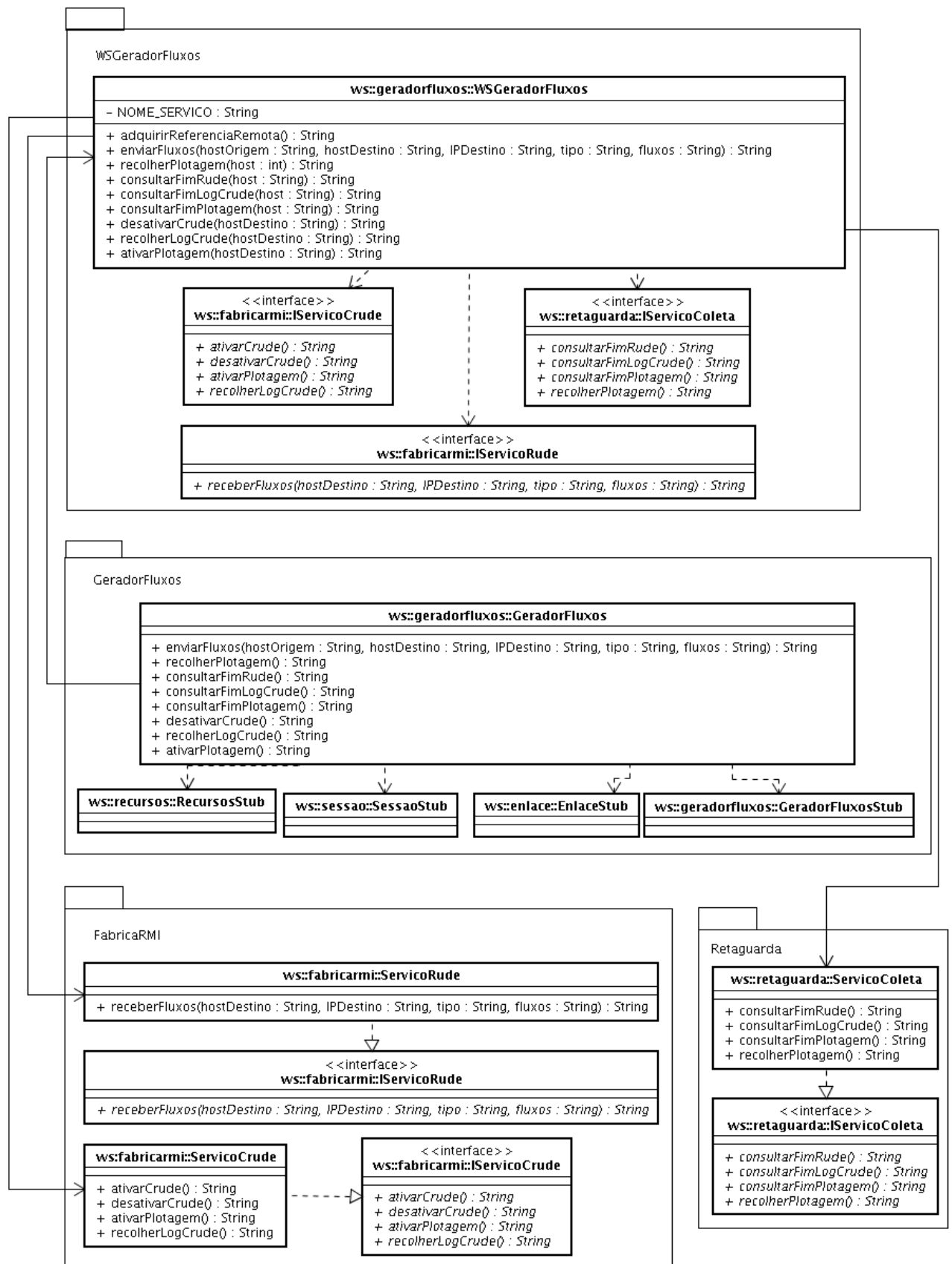


Fig. 4.14: Diagrama de Classes para o Experimento Gerador de Fluxos.

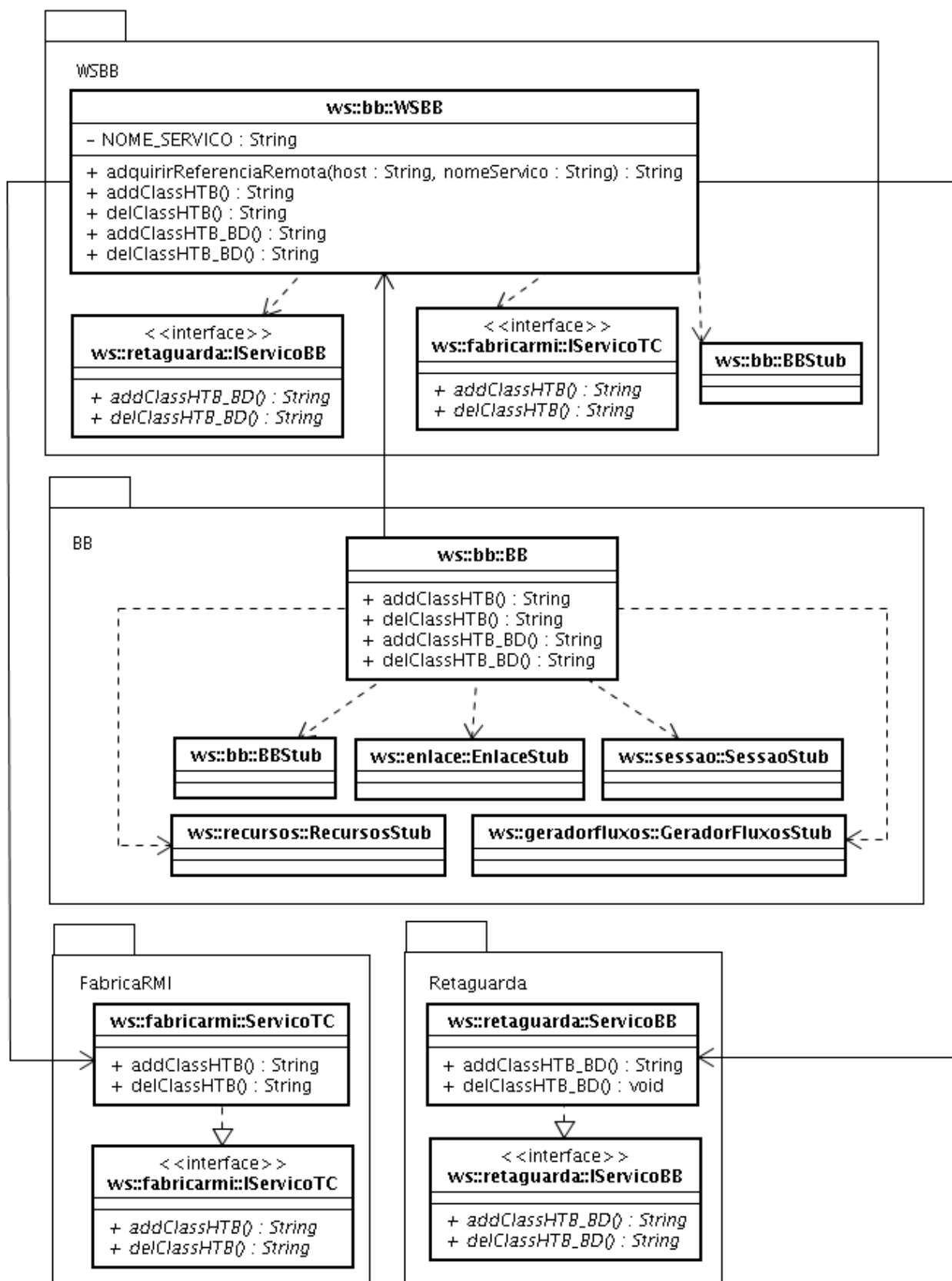


Fig. 4.15: Diagrama de Classes para o Experimento BB.

Capítulo 5

Implementação e Resultados

Este capítulo apresenta a implementação da arquitetura *DiffServ* no NetLab WebLab e apresenta experimentos de geração de fluxos reais e simulados no domínio do laboratório. Os aplicativos Web de Gerência Administrativa e de Uso do NetLab WebLab são fundamentais para garantir a instanci- ação adequada de recursos físicos e lógicos para os experimentos. A implementação da arquitetura de *software* do WebLab possui um fraco acoplamento com a arquitetura física do laboratório e isso permite que aplicativos mais elaborados como um *Bandwidth Broker* para experimentos *DiffServ* seja implementado sem alterar a estrutura física do domínio. A implementação do BB com suporte a monitoramento e readaptação de fluxos também será descrita nesse capítulo.

As características inerentes do ambiente *DiffServ* permitem estudar o comportamento de diferen- tes tipos de fluxos e avaliar os resultados nos diferentes cenários oferecidos. O WebLab prioriza o desempenho da comunicação ao utilizar RMI entre os elementos da rede interna e ao simplificar os mecanismos de encaminhamento de pacotes SOAP; promove escalabilidade ao reduzir o acoplamento entre a arquitetura física e a arquitetura de *software*; promove a segurança no acesso às informações com o uso de serviços de acesso e utiliza arquiteturas independentes do sistema operacional.

5.1 Disponibilização de Experimentos no NetLab WebLab

Os experimentos do NetLab WebLab são disponibilizados como aplicações Web no *host* servidor do laboratório. Eles estão fracamente acoplados à infraestrutura de *hardware* porque são formados de acordo com o modelo de blocos que distribui os elementos de requisição, comunicação e interação direta com os dispositivos gerenciáveis. Apenas o último bloco interage diretamente com o *host* gerenciável e isso permite a manutenção distribuída dos componentes de cada bloco. Os experimentos também estão fracamente acoplados à infraestrutura dos aplicativos Web de Gerência Administrativa e de Uso do WebLab porque são desenvolvidos independentes deles.

Um experimento é formado por um conjunto de recursos físicos e lógicos. Os recursos físicos são formados pelos *hosts* e seus respectivos sub-recursos, como interfaces de rede, por exemplo. Os recursos lógicos são os aplicativos incluídos no bloco servidor que atuam diretamente nos *hosts* do laboratório. O modelo distribuído permite a criação de diversos recursos lógicos independentes do experimento. Dessa forma, um ou mais experimentos podem ser formados pela composição de diversos recursos lógicos. Da mesma forma, um experimento pode ser disponibilizado com diversos recursos físicos.

O usuário tem acesso a um experimento através do *site* da aplicação Web de gerência de uso do experimentos, como ilustrado na Fig. 4.2. Apenas são vistos os experimentos disponibilizados para o WebLab. Essa disponibilização ocorre no *site* da aplicação Web de gerência administrativa, como ilustrado na Fig. 4.2. Um experimento precisa ser cadastrado antes de ser disponibilizado para um ou

mais WebLabs. Quando o usuário clica no *link* do experimento, no *site* do laboratório, é iniciado o processo de *download* da aplicação com o uso da tecnologia *Java Web Start* [74]. O usuário deve ter uma plataforma *Java Runtime Environment* (JRE) [78], versão 1.2.2 ou mais recente instalada em seu *host* para acessar o experimento.

O aplicativo Web do laboratório verifica as credenciais do usuário e a disponibilidade de uso do experimento baseada no prévio agendamento. A identificação do experimento é coletada a partir do *link* do experimento cadastrado e as credenciais do usuário são coletadas com um serviço de acesso. Caso as verificações retornem uma condição válida, o *servlet ServletExp* faz a coleta na base de dados de todos os recursos associados ao experimento. Depois, esses recursos são separados em dois grupos: recursos físicos e lógicos. Para cada recurso físico são instanciados os recursos lógicos do experimento. O critério de uso ou não desses elementos irá depender de como o experimento será utilizado. O *ServletExp* apenas garante que eles serão ativados pelas fábricas de objetos antes do aplicativo JWS ser iniciado remotamente. O *servlet* continua o seu processamento e prepara os objetos servidores de Retaguarda de acordo com a necessidade do experimento. Esses serviços irão iniciar os objetos instanciados pela fábrica ou ficarão à disposição para quaisquer outras necessidades de interação do aplicativo JWS com o servidor (por exemplo, consulta à base de dados do WebLab). Caso ocorra algum problema ou exceção em qualquer uma dessas etapas, o aplicativo não é iniciado.

Para permitir o início do *download* o *ServletExp* encaminha para o arquivo *iniciarExperimento.jsp* do respectivo projeto do experimento apenas o nome do experimento, o usuário e os *hosts*, como mostrado no trecho a seguir:

```
String URL = "http://" + enderecoIPServidor + ":" + porta + "/" +
    nomeExperimento + "/iniciarExperimento.jsp?" +
    "usuario=" + IDUsuario + "&hosts=";
```

O arquivo *iniciarExperimento.jsp* é um arquivo JSP do tipo JNLP (*Java Network Launching Protocol*) que recebe os argumentos do *servlet* e inicia a aplicação JWS.

5.1.1 Coleta e Representação Virtual de Recursos

Para diminuir a quantidade de parâmetros informados para a URL do arquivo JSP, os demais argumentos podem ser solicitados diretamente pelo aplicativo JWS a partir dos objetos instanciados pela fábrica de Retaguarda. Como exemplo, para a recuperação de sub-recursos de um recurso (NICs, por exemplo) e os enlaces entre os *hosts*, primeiro é necessário que essas informações sejam registradas no aplicativo de Gerência Administrativa, como ilustrado na Fig. 5.1.



Fig. 5.1: Tags para Indicar os Sub-recursos (NICs) de Cada Recurso (*host*).

O aplicativo JWS envia para o *Web Service Recursos* a solicitação para a recuperação dos sub-recursos de cada um dos recursos físicos do experimento. O *Web Service Recursos* encaminha a requisição para o *objeto servidor Recursos* instanciado pela fábrica de Retaguarda. Esse objeto recupera da base de dados as informações dos sub-recursos registrados para o recurso do experimento.

Da mesma forma, o aplicativo JWS solicita as informações sobre os enlaces entre os *hosts*. As informações sobre os enlaces também são registradas nos aplicativos Web de Gerência Administrativa, como ilustrado na Fig. 5.2.



Fig. 5.2: Tags para Indicar os Enlaces entre as NICs dos *Hosts*.

De posse dessas informações, o aplicativo JWS é iniciado no *host* do usuário. O aplicativo ilustra virtualmente a estrutura lógica do experimento utilizando a biblioteca gráfica JGraph [79] que possui compatibilidade com o padrão Swing Java.

5.1.2 Infraestrutura do NetLab WebLab

A Fig. 5.3 ilustra a rede física do laboratório. Os *hosts* foram configurados para desempenhar o papel de roteadores de borda e de núcleo da rede. Os *hosts* HODES e ZEUS foram reservados como servidores do laboratório e não fazem parte dos experimentos. Estes *hosts* estão associados a endereços IP distintos para permitir a negociação de recursos entre domínios diferentes. A Tab. 5.1 mostra as características do *hardware* dos computadores do laboratório e a quantidade de NICs 10/100Mbps e 10/100/1000Mbps por *host*.

Host	CPU / cache	RAM	HD	# 100Mbps	# 1000Mbps
Hodes	Intel Core 2 Duo, 2.4Ghz / 4096KB	3GB	250GB	2	1
Helios	Intel Pentium 4, 3GHz / 512KB	1GB	40GB	1	3
Gaia	Intel Pentium 3, 600MHz / 256KB	256MB	40GB	1	2
Urano	Intel Pentium 3, 600MHz / 256KB	256MB	40GB	1	2
Poseidon	Intel Pentium 3, 600MHz / 256KB	256MB	40GB	1	3
Zeus	Intel Pentium 4 HT, 3GHz / 1024KB	512MB	80GB	2	1

Tab. 5.1: Características dos computadores do NetLab WebLab.

Os *hosts* HELIOS, GAIA, POSEIDON e URANO, e suas respectivas interfaces de rede, são os recursos físicos do laboratório. Cada um desses *hosts* possui interfaces de rede Ethernet gigabit e estão conectados a um *switch* Ethernet gigabit de 16 portas. Os *hosts* do laboratório também estão conectados a uma rede de retaguarda através de um *hub* Ethernet 10Base-T. Essa rede garante a conectividade entre os *hosts* e é através dela que são realizados o início e o término consistente dos

experimentos da rede. Os computadores do WebLab utilizam o sistema operacional *Slackware* 10.2 e *kernel* 2.6.15.6 com suporte a *DiffServ*.

A interface de rede para a rede de retaguarda não está disponível para manipulação pelos experimentos. A rede física, associada a diferentes VLANs no *switch*, juntamente com a configuração das rotas nos *hosts*, permite a formação da rede lógica para os experimentos, ilustrada na Fig. 5.4.

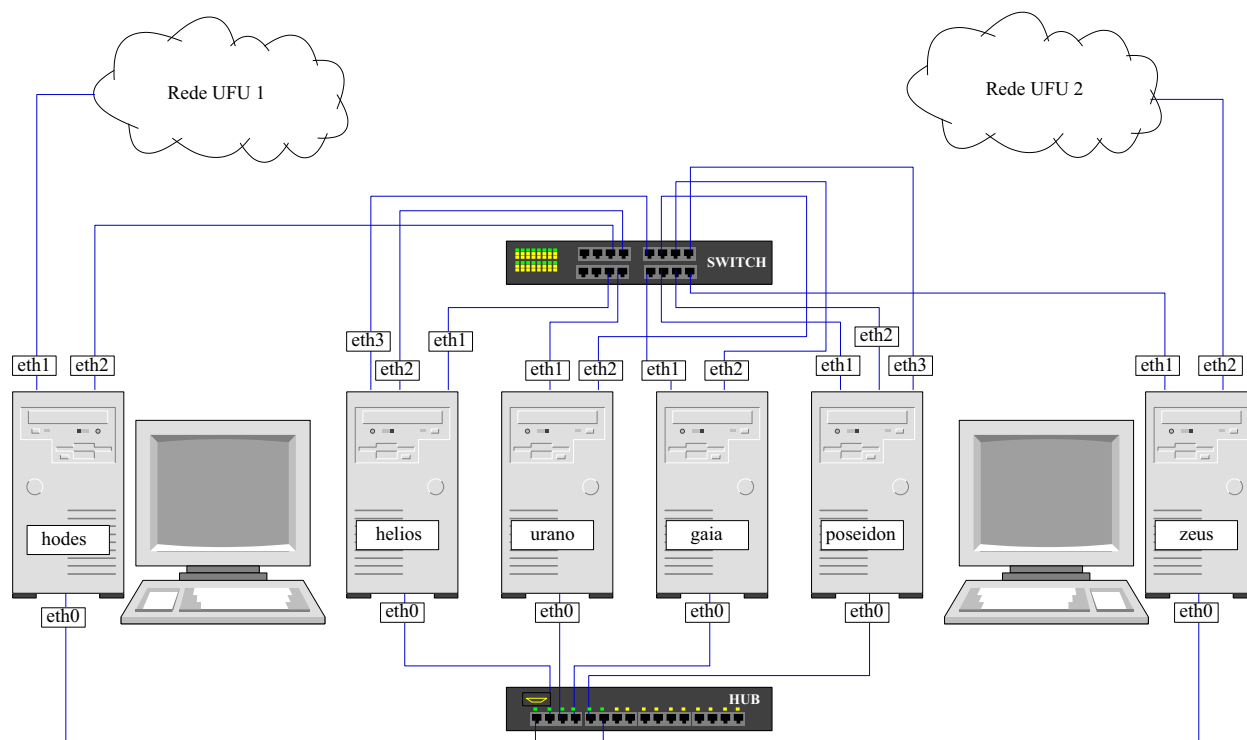


Fig. 5.3: Rede Física do Laboratório.

A infraestrutura de *software* do laboratório é composta por um Web *container* Apache Tomcat com serviços Web Axis2/Java, um banco de dados relacional MySQL e utiliza a tecnologia Java RMI para realizar a comunicação entre a rede interna do WebLab e o *container* de Web Services. Os hosts *HODES* e *ZEUS* mantêm os Web Servers Tomcat no laboratório e o *container* para Web Services Axis2. Dessa forma, o NetLab WebLab pode ser acessado por meio de dois endereços IP distintos.

5.2 Controle de Tráfego no Domínio *DiffServ*

A Internet atual não faz distinção entre diferentes tipos de tráfego [80]. A distinção de tráfego traz a capacidade dos ISPs oferecerem mais ou menos recursos de rede de acordo com o perfil dos usuários e das necessidades das aplicações.

Com a disseminação exponencial da Internet, uma grande quantidade de serviços têm contribuído para aumentar o congestionamento na rede. A maioria desses serviços contempla mídias contínuas como áudio e vídeo, altamente sensíveis ao estado da rede, que demandam garantias de desempenho quanto à qualidade de serviço. Dentre as várias propostas para a provisão de QoS na Internet, destaca-se a arquitetura *DiffServ*. As deficiências dessa arquitetura motivaram extensões à mesma para acomodar a negociação global e o monitoramento da alocação de recursos com o uso de experimentos no NetLab WebLab.

A necessidade de garantia de recursos para aplicações tempo-real que utilizam a Internet como meio de integração de serviços de telefonia fixa, difusão de fluxos de áudio e vídeo, laboratórios

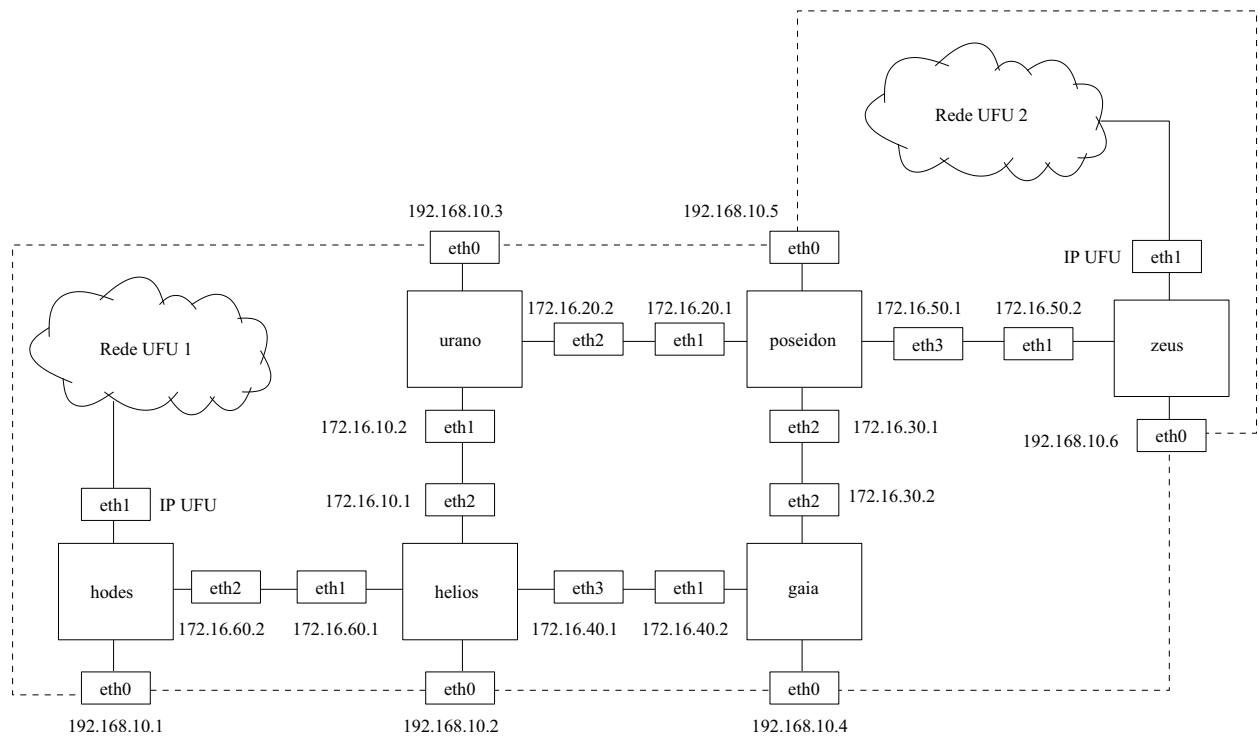


Fig. 5.4: Rede Lógica do Laboratório.

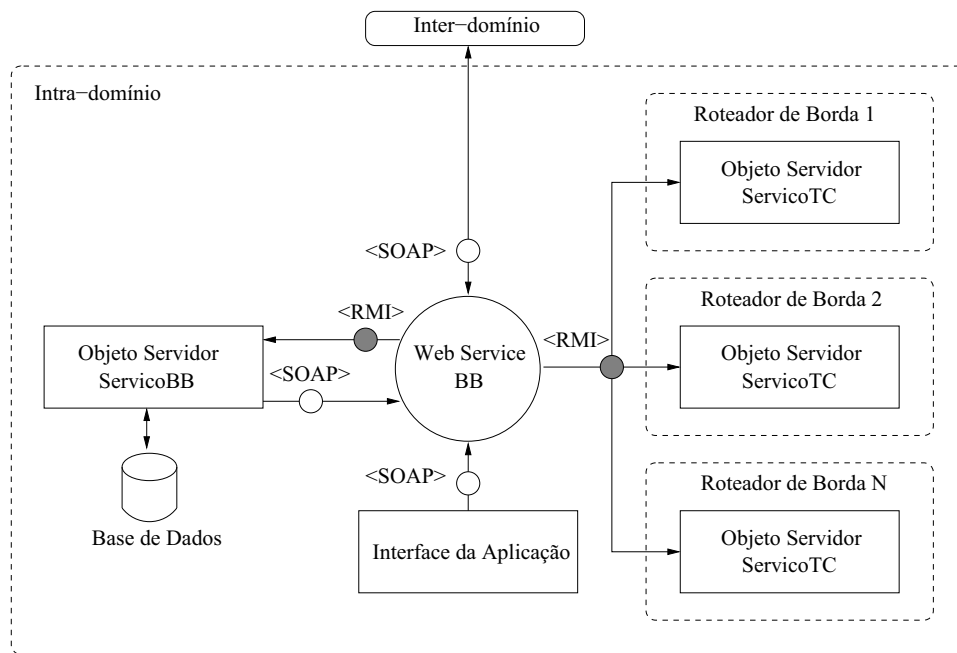
virtuais, entre outros, exige um nível maior de qualidade de envio e recepção de informações se comparado com aplicações convencionais. A pilha TCP/IP não assume garantias quanto à vazão, variação do atraso (*jitter*), perda de pacotes e/ou taxa de erros.

Apesar de *DiffServ* ser capaz de oferecer QoS diferenciada para as aplicações, muitas questões para a implementação da arquitetura ainda não foram resolvidas. Dentre elas, não é especificado como o SLA para a reserva de recursos da rede deve ser estabelecido entre o usuário da aplicação e o domínio *DiffServ*, como realizar um controle de admissão eficiente para redes sem conexão, como negociar a reserva de recursos entre domínios, como realizar o agendamento da reserva de recursos, como suportar a reserva de recursos *multicast DiffServ*, entre outros [3].

Os experimentos *DiffServ* no WebLab foram desenvolvidos para estudar a abordagem *DiffServ* e resolver parte das limitações dessa arquitetura. Dois experimentos foram implementados: o experimento *Bandwidth Broker* que promove a negociação global de recursos com o uso de SLAs; o experimento Gerador de Fluxos que exige recursos de QoS para ser executado, o qual é utilizado para verificar o funcionamento do experimento anterior.

5.3 Experimento *Bandwidth Broker*

O experimento *Bandwidth Broker* implementa o agente centralizador do domínio *DiffServ* e permite ao usuário do WebLab criar configurações *DiffServ* para serem utilizadas pelos experimentos que suportam o controle de tráfego. Além disso, o aplicativo auxilia no entendimento da negociação de recursos intra-domínio. O experimento *Bandwidth Broker* faz a negociação de recursos antes do início do experimento, insere as políticas de controle de tráfego apenas nos roteadores de borda dos recursos informados e remove as configurações ao final da experimentação. O experimento *Bandwidth Broker* é uma implementação do agente de redes que gerencia a alocação e o uso de recursos dentro e fora do domínio *DiffServ* [24]. A Fig. 5.5 ilustra a arquitetura desse experimento.

Fig. 5.5: Arquitetura do Experimento *Bandwidth Broker*.

A Interface da Aplicação do BB permite o gerenciamento do domínio *DiffServ*. Essa aplicação exibe as informações sobre SLA, PHB, PHB atribuído para o experimento, roteadores de borda do experimento, largura de banda da rede, BAR, RAR e configurações *DiffServ*.

O experimento BB permite criar a configuração *DiffServ* conforme mostra a Fig. 5.6. Podem ser criadas disciplinas de fila, classes e filtros através de um ambiente gráfico em que o usuário informa apenas os parâmetros de cada comando *tc* do pacote IPROUTE2 [68]. O aplicativo acompanha a estrutura da configuração *DiffServ* e apresenta uma estrutura de árvore porque os comandos *DiffServ* são dependentes uns dos outros e devem ser inseridos respeitando a hierarquia de formação de cada PHB. Os parâmetros *DiffServ* são enviados pelo BB para o *host* da rede interna do laboratório. Cada *host* possui um *objeto servidor servicoTC* capaz de interpretar esses parâmetros, montar o comando *tc* e executá-lo. O trecho que segue exibe os comandos para controle do tráfego de pacotes IP gerados pelo objeto servidor no *host* do WebLab. A criação dessa configuração é simplificada com o experimento BB, como mostra a Fig. 5.6, onde apenas os parâmetros de cada comando são informados e organizados em tabelas.

```
# -- AF1 [5MB]
#1 -- qdisc GRED para AF1
    tc qdisc add dev eth0 parent 2:10 gred setup DPs 4 default 2 prio
#2 -- qdisc gred AF 1 DP (Drop precedence) 1
    tc qdisc change dev eth0 parent 2:10 gred limit 60KB min 20KB \
    max 55KB burst 40 avpkt 1472 bandwidth 5mbps DP 1 \
    probability 0.02 prio 2
#3 -- qdisc GRED AF 1 DP 2
    tc qdisc change dev eth0 parent 2:10 gred limit 50KB min 15KB \
    max 45KB burst 30 avpkt 1472 bandwidth 5mbps DP 2 \
    probability 0.04 prio 3
#4 -- qdisc GRED AF 1 DP 3
    tc qdisc change dev eth0 parent 2:10 gred limit 40KB min 5KB \
    max 35KB burst 20 avpkt 1472 bandwidth 5mbps DP 3 \
    probability 0.06 prio 4
```

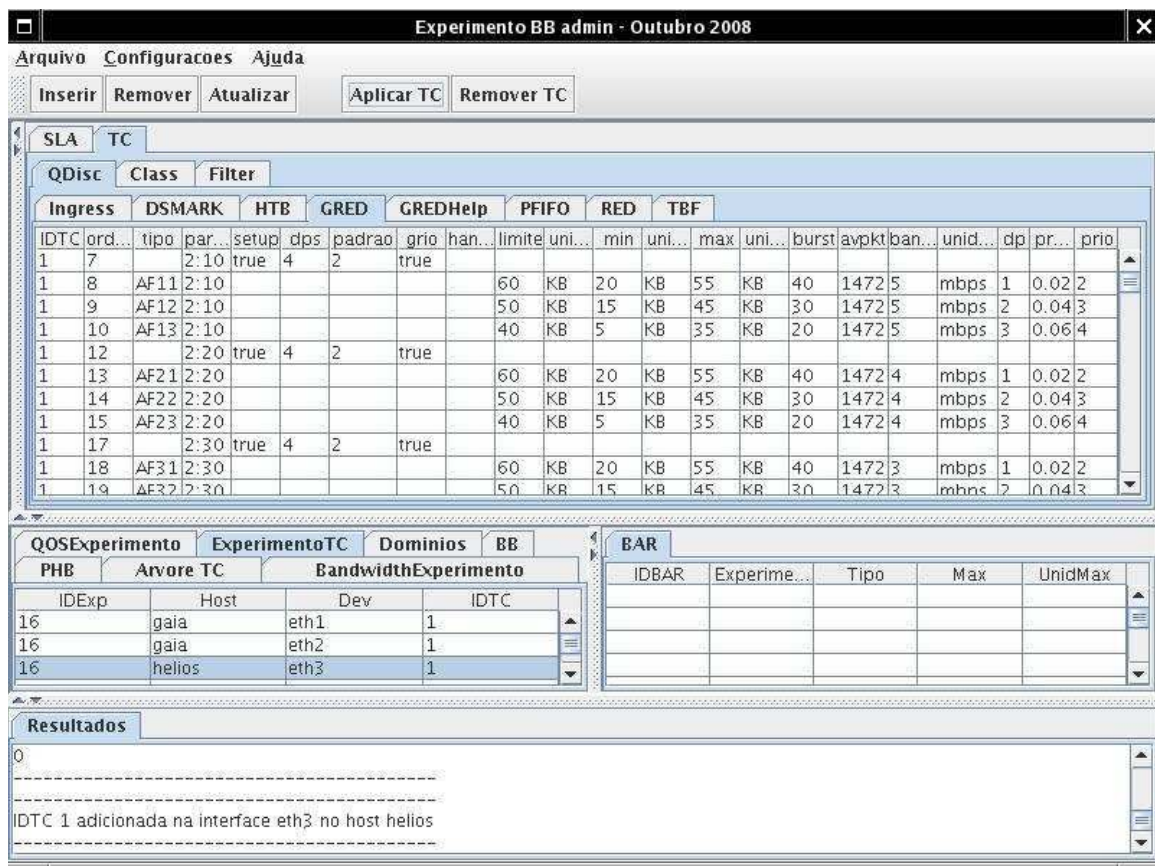


Fig. 5.6: Experimento BB no NetLab WebLab.

A representação em árvore é outra alternativa criada no experimento BB para simplificar o acompanhamento da configuração *DiffServ*, como ilustrado a seguir:

```

1 .
2 |-- qdisc_ingress_ffff:
3 |   |-- filter_u32_AF11
4 |   |-- filter_u32_AF12
5 |   |-- filter_u32_AF13
6 |
7 ...
15 |   |-- filter_u32_BE
16 |   |-- filter_u32_EF
17 |-- root
18   |-- qdisc_dsmark_1:0
19       |-- qdisc_htb_2:0
20           |-- class_htb_2:1
21               |-- class_htb_2:10
22                   |-- qdisc_gred_AF11
23                   |-- qdisc_gred_AF12
24                   |-- qdisc_gred_AF13
25 |
26 ...
37 |   |-- class_htb_2:50
38 |       |-- qdisc_red_BE
39 |-- class_htb_2:60
40     |-- qdisc_pfifo_EF

```

5.3.1 Base de Dados

A atuação do BB depende das informações armazenadas na base de dados. Para os experimentos de rede *DiffServ*, a cada nova solicitação de inicialização de um experimento, o BB é consultado para verificar se há recursos disponíveis, ou seja, largura de banda disponível para uma nova alocação. Isso é necessário para evitar que um experimento seja iniciado sem a quantidade de recursos mínima para o seu funcionamento adequado. Essa solução evita a competição de largura de banda no momento da submissão de fluxos à rede por dois ou mais experimentos. Dentro do experimento, a submissão de fluxos respeita as restrições de controle de tráfego: os fluxos que excederem os valores acordados terão os seus pacotes descartados.

Para um experimento podem ser atribuídos um ou mais PHBs. O BB controla quais PHBs podem ser atribuídos para um experimento de acordo com a largura de banda alocada para o PHB.

Cada PHB possui uma quantidade mínima de largura de banda. O BB controla essa alocação verificando se a soma das porções de largura de banda alocadas para os PHBs é menor ou igual à capacidade do enlace no laboratório. Por exemplo, para um PHB AF1 será atribuída uma quantidade de largura de banda subtraída da largura de banda global do enlace no domínio do WebLab. No entanto, podem ser criadas diferentes implementações de controle de tráfego para AF1.

O BB mantém o SLA para experimentos porque as restrições de recursos devem respeitar os requisitos mínimos para a execução adequada do experimento. Cada linha do SLA representa um SLS com as informações do usuário, experimento, parâmetros de QoS e período de utilização reservado para o usuário.

Tabela PHB

Campo	Descrição
PHB (chave primária)	Serviço de encaminhamento de pacotes
Min	Valor da largura de banda mínima
UnidMin	Unidade da largura de banda mínima (Mbps, Kbps, etc.)
Max	Valor da largura de banda máxima
UnidMax	Unidade da largura de banda máxima (Mbps, Kbps, etc.)
Alocado	Quantidade de largura de banda alocada (Mbps, Kbps, etc.)
Ativo	Quantidade de experimentos que utilizam o PHB
UsoReal	Max / Ativo

Tab. 5.2: Tabela PHB.

O campo *PHB* pode ter os seguintes valores: EF, BE, AF1, AF11, AF12, AF13, ..., AF43. O somatório dos valores inseridos na coluna *Max* deve ser menor ou igual à largura de banda do domínio.

O somatório dos valores inseridos na coluna *Min* deve ser menor ou igual à largura de banda do PHB. Exemplo: o somatório dos valores de AF11 e AF12 deve ser menor ou igual ao valor de AF1. O valor da coluna *Max* para esses casos deve ser menor ou igual ao valor *Max* de AF1.

A coluna *Alocado* indica a largura de banda que foi alocada para um ou mais experimentos ativos. A coluna *Ativo* indica a quantidade de experimentos ativos. A coluna *UsoReal* indica a quantidade de recursos distribuída entre os experimentos.

Tabela PHB_EXPERIMENTO

Essa tabela define quais PHBs da tabela *PHB* estão associados aos experimentos. Um PHB pode estar associado a mais de um experimento. O campo *Solicitado* é utilizado para requisitar uma porção

de recursos do PHB específico.

Campo	Descrição
IDExp	Identificação do experimento
PHB (chave estrangeira)	Serviço de encaminhamento de pacotes
Solicitado	Quantidade de recursos solicitados do PHB
UnidSolicitado	Unidade da quantidade de recursos solicitados do PHB

Tab. 5.3: Tabela PHB_EXPERIMENTO.

Tabela SLA

Campo	Descrição
IDSLA (chave primária)	Identificação do SLA
IDUsuario	Identificação do usuário
IDExp	Identificação do experimento
QOS	Lista de PHB com largura de banda mínima e máxima
DataInicio	Data de início reservada para o experimento
HoraInicio	Hora de início reservada para o experimento
DataFim	Data de término do uso do experimento
HoraFim	Hora de término do uso do experimento

Tab. 5.4: Tabela SLA.

Cada tupla da tabela SLA define um SLS, ou seja, os aspectos técnicos do SLA.

A coluna QOS é utilizada na negociação de recursos entre os BBs adjacentes. A lista é uma string com o seguinte formato:

```
PHBn,DS Field,largura de banda mínima do PHBn,largura de banda máxima do PHBn;
```

Os limites mínimos e máximos da largura de banda de cada PHB são recuperados da tabela PHB porque esses limites foram verificados pelo BB. Como exemplo:

```
AF11, 0x28, 1, mbps, 3, mbps; AF21, 0x48, 1, mbps, 2, mbps;
AF22, 0x50, 1, mbps, 1, mbps; BE, 0x0, 1, mbps, 1, mbps;
EF, 0xb8, 1, mbps, 3, mbps;
```

O tempo de utilização de um experimento também é definido no SLA.

Tabela RAR

A tabela RAR é utilizada para armazenar as informações sobre a requisição da alocação de recursos por um experimento. No entanto, essa tabela também pode ser utilizada para fins de *log* porque são gravados os *status* (aceite/recusa) das requisições ao BB.

A coluna QOS indica a largura de banda que pôde ser alocada para a requisição, ou seja, uma quantidade menor de largura de banda pode ser atribuída ao PHB do experimento, com base nas informações do SLA.

O período de uso do experimento deve estar dentro do intervalo informado no SLA para que o BB aceite a requisição.

Campo	Descrição
IDRAR	Identificação da RAR
IDUsuario	Identificação do usuário
IDExp (chave estrangeira)	Identificação do experimento
IDSLA (chave estrangeira)	Identificação do SLA
QOS	Lista com a largura de banda alocada para cada PHB
DataInicio	Data de início solicitada para o experimento
HoraInicio	Hora de início solicitada para o experimento
DataFim	Data de término do uso do experimento
HoraFim	Hora de término do uso do experimento

Tab. 5.5: Tabela RAR.

Campo	Descrição
IDExp (chave primária)	Identificação do experimento
Host (chave primária)	Roteador de borda
Dev (chave primária)	Interface de rede do roteador de borda
IDTC	Identificação do grupo de configurações tc

Tab. 5.6: Tabela EXPERIMENTO_TC.

Tabela EXPERIMENTO_TC

Cada experimento pode ter uma configuração de disciplinas de fila diferente dos demais. Isso é necessário porque existem experimentos que possuem uma ou mais NICs que exigem configurações diferentes para os fluxos de entrada e saída do *host*. As NICs podem ter diferentes configurações de disciplinas de fila. Então, a coluna *IDTC* mantém um número que identifica qual configuração será atribuída para a NIC do *host*. Isso confere maior escalabilidade para a atribuição de configurações: um mesmo conjunto de configurações de tráfego que possui um IDTC único pode ser atribuído a uma ou mais NICs em diferentes *hosts* de diferentes experimentos, sem a necessidade de criar uma nova configuração para cada NIC em outro experimento.

Tabelas para o Controle de Tráfego

As tabelas para o controle de tráfego são utilizadas para armazenar as configurações *DiffServ* proporcionadas pela ferramenta *tc* [68]. Uma vez atribuídas, a recuperação de quais configurações estão presentes no *host* não é trivial porque a ferramenta *tc* não oferece suporte adequado para a recuperação dessas informações. Outra informação não suportada é a ordem em que foram inseridas. Essas informações são relevantes porque a ordem de inserção e remoção das configurações respeita uma estrutura de dados hierárquica do tipo árvore.

Para resolver esse problema foram criados dois campos nas tabelas que armazenam as configurações *DiffServ*. O campo *IDTC* é utilizado para agrupar um conjunto de configurações de controle de tráfego. Por exemplo, o *IDTC* 1 pode aparecer em uma ou mais tabelas. Isso permite agrupar as configurações com o mesmo *IDTC*. O campo *Ordem* é utilizado juntamente com o *IDTC* para definir a ordem dos comandos inseridos, porque essa ordem não é definida pela ferramenta *tc*. O campo *Tipo* é opcional e é utilizado para tratar as restrições impostas pela tabela *PHB*. Por exemplo, caso a tabela *PHB* defina para AF1 os limites de largura de banda mínima e máxima de 10MB e 20MB, respectivamente, e caso seja atribuído AF1 para uma tupla da classe HTB, essa tupla deve respeitar os limites AF1 da tabela *PHB*. Os demais campos seguem a sintaxe do co-

mando. As seguintes tabelas foram criadas: *FILTER_TCINDEX*, *FILTER_U32*, *QDISC_DSMARK*, *QDISC_GRED*, *QDISC_HTB*, *QDISC_INGRESS*, *QDISC_PFIFO*, *QDISC_RED*. As tabelas seguem o mesmo exemplo da Tab. 5.7 com a alteração dos campos da sintaxe de cada disciplina de fila, classe e filtro.

Campo	Descrição
IDTC	Identificação do grupo de configurações tc
Ordem	Ordem do comando no grupo
Tipo	Associação do Tipo ao comando (Ex.: AF1, AF2, etc.)
Parent	Indica a classful qdisc pai
ClassId	Identificação da classe
Rate	Taxa de encaminhamento de pacotes
UnidRate	Unidade da taxa (mbps, kbps, etc.)
Ceil	Limite superior da taxa de encaminhamento de pacotes
UnidCeil	Unidade da taxa superior (mbps, kbps, etc.)

Tab. 5.7: Tabela *CLASS_HTB*.

Tabela BB

Campo	Descrição
IP	Endereço IP do BB.
Bandwidth	Largura de banda do domínio
UnidBandwidth	Unidade da largura de banda (gbps, mbps, etc.)
Disponivel	Largura de banda disponível para outros experimentos
UnidDisponivel	Unidade da largura de banda disponível (gbps, mbps, etc.)

Tab. 5.8: Tabela BB.

Na tabela BB, o campo *bandwidth* é utilizada para definir o limite para a atribuição de valores de largura de banda para os PHBs. O campo *Disponivel* é utilizado para determinar a quantidade de largura de banda disponível para iniciar novos experimentos e respeita a seguinte fórmula:

$$\text{Disponivel} = \text{largura de banda do BB} - \text{largura de banda do experimento iniciado}$$

O BB atua na verificação dos limites de largura de banda a serem alocados, reserva de banda e atribuição das configurações de controle de tráfego para os experimentos. Apenas um usuário pode interagir com um experimento de redes por vez para evitar o uso dos mesmos recursos físicos. Dois ou mais usuários podem utilizar o laboratório ao mesmo tempo, mas apenas com experimentos que controlam recursos diferentes. Esse controle fica a cargo da gerência de reserva de experimentos que também realiza operações de persistência. Fica a cargo do administrador do WebLab verificar se o uso concorrente de dois ou mais experimentos interfere no resultado da experimentação.

5.3.2 Web Service BB

O elemento *Web Service BB* promove a integração interna e externa da arquitetura, e realiza as operações de roteamento das informações para dentro e fora do domínio. O roteamento interno ocorre nas seguintes situações:

- quando uma RAR é solicitada pela *interface da aplicação*, o *Web Service BB* deve rotear a mensagem para o *objeto servidor ServicoBB*;
- caso a RAR seja aceita, o *Web Service BB* deve encaminhar as configurações *DiffServ* para o *objeto servidor ServicoTC* no respectivo roteador de borda do experimento;
- caso a RAR não seja aceita pelo *objeto servidor ServicoBB*, o *Web Service BB* deve rotear a mensagem para a *interface da aplicação*;
- quando uma solicitação de operação na base de dados é solicitada pela *interface da aplicação* o *Web Service BB* deve rotear a mensagem para o *objeto servidor ServicoBB*. Este deve verificar a consistência das informações e decidir se aceita ou não a requisição.
- caso a operação seja aceita ou não, o *Web Service* deve encaminhar o resultado para a *interface da aplicação*.

5.3.3 Objeto Servidor ServicoBB

O *objeto servidor ServicoBB* implementa a lógica para o gerenciamento da alocação e uso dos recursos *DiffServ* dentro e fora do domínio. Ele mantém as informações sobre a alocação atual de recursos e interpreta as novas requisições por meio de consultas à base de dados. Esse objeto informa qual marcação o tráfego de um experimento deve receber de acordo com o SLA. A negociação de recursos é realizada de forma automatizada dentro do domínio respeitando as regras para a inserção de valores nas tabelas. Para a negociação de recursos fora do domínio é utilizado o *Web Service BB* para encaminhar as requisições aos BBs adjacentes.

O *objeto servidor ServicoBB* é um objeto instanciado pela fábrica de objetos de Retaguarda que realiza operações de persistência no *host* servidor do WebLab. Além de realizar consultas à base de dados, esse elemento da arquitetura do BB armazena as operações de inserção, remoção e atualização das configurações *DiffServ* que são realizadas nos roteadores de borda do experimento. A armazenagem dessas operações depende da verificação dos limites impostos aos valores das tabelas.

Para requisições RAR, o *objeto servidor BB* verifica o SLA do experimento. Caso existam recursos disponíveis para a alocação, o objeto encaminha para o *Web Service BB* a solicitação para preparação das configurações de controle de tráfego nos roteadores de borda do experimento. Caso não haja recursos disponíveis para a alocação, uma mensagem de advertência é retornada para a *interface da aplicação*.

Heurísticas para Reserva de Recursos

O *objeto servidor ServicoBB* implementa um conjunto de heurísticas que reduzem a competição de recursos entre experimentos e otimizam o uso da largura de banda. A Tab. 5.9 apresenta um exemplo de distribuição de banda com base nas heurísticas do BB.

A concorrência que ocorre entre fluxos reduz a qualidade de encaminhamento de pacotes, resultando em maior atraso na entrega, perda de pacotes e aumento do *jitter*. Entre fluxos de mesmo PHB é desejado que exista uma distribuição equitativa entre eles de forma que cada fluxo possua uma quantidade mínima de banda. Entre fluxos de diferentes PHBs é desejado que exista a garantia mínima de banda sem que essa garantia implique na penalização excessiva de fluxos de outros PHBs. A garantia de recursos é uma garantia estatística, ou seja, em virtude da capacidade do enlace será realizado um tratamento diferenciado do tráfego com a distribuição dos recursos entre os diferentes tipos de serviços de encaminhamento. Diante dessas considerações, as seguintes heurísticas foram propostas:

PHB	Min (%)	Max (%)
AF1	1	20
AF2	1	20
AF3	1	20
AF4	1	10
EF	5	20
BE	0.1	10

Tab. 5.9: Exemplo de distribuição de banda entre os PHBs.

1. $\sum_{i=1}^n \max_{PHB} i \leq bandwidth$: Cada PHB possui uma configuração com determinada quantidade de banda do enlace. Quando a banda estiver com baixa utilização faz sentido permitir que os fluxos utilizem mais recursos da rede. No entanto, se muitos fluxos forem admitidos, as limitações impostas por disciplinas de fila hierárquicas, tais como HTB (*Hierarchical Token Bucket*) não são suficientes para descartar adequadamente os pacotes inadimplentes de forma a promover uma correta preempção de recursos, com perdas reduzidas, para os fluxos de maior prioridade, porque os fluxos já foram admitidos. Ou seja, para alguns poucos PHBs de alta prioridade, os fluxos de menor prioridade serão severamente penalizados e podem nem mesmo conseguir uma quantidade mínima de recursos. O inverso também é válido: quando muitos fluxos de baixa prioridade são admitidos estes serão severamente penalizados para permitir a garantia mínima de recursos para alguns poucos fluxos de alta prioridade. Essa regra garante que cada PHB possua uma configuração com determinada porção da rede, ou seja, a máxima porção do enlace que pode ser alocada para cada PHB ($\sum_{i=1}^n \max_{PHB} i$) deve ser menor do que a capacidade do enlace global (*bandwidth*). Essa regra limita a quantidade de fluxos logo na entrada do domínio, sem interferir nas demais regras de distribuição de recursos promovidos pelas demais configurações DiffServ, sem penalizar outros fluxos ou ultrapassar a capacidade do enlace.
2. $\sum_{i=1}^n \text{alocado}_{PHB} i \leq \max_{PHB}$: Cada configuração DiffServ para um PHB é capaz de definir os limites máximos de uso de porções do enlace ($\max_{PHB}$). Quando um fluxo é admitido no domínio ele solicita determinada quantidade de recursos ao BB. Como fluxos agregados são mapeados para PHBs faz sentido que a configuração de um PHB possa admitir tantos fluxos mapeados quanto possíveis, desde que o somatório dos recursos do enlace alocados para cada admissão sejam mantidos dentro da porção do PHB ($\sum_{i=1}^n \text{alocado}_{PHB} i$).
3. $\min_{PHB} \leq \text{alocado}_{PHB}$: Quando uma parcela de recursos do enlace é alocada para um fluxo mapeado para um PHB (alocado_{PHB}) deve existir a garantia de que uma quantidade mínima de recursos ($\min_{PHB}$) será mantida para a alocação. No entanto, à medida que o enlace é saturado e mais fluxos são admitidos, cada um deles deve possuir uma quantidade mínima de recursos que justifique a alocação. Para os fluxos de mesmo PHB isso significa que os recursos serão compartilhados igualmente entre os fluxos. Essa garantia implica na determinação da quantidade de fluxos que podem ser submetidos ao mesmo tempo com uma quantidade mínima de recursos.

As regras são divididas em dois grupos: a primeira regra mantém o critério para a definição dos limites de largura de banda; as demais mantêm os critérios para a admissão dos fluxos na rede. Como consequência, $\sum_{i=1}^n \text{alocado}_{PHB} i \leq bandwidth$, ou seja, cada PHB admite uma determinada quantidade de fluxos com recursos mínimos garantidos para cada um desses fluxos. Quando a porção do enlace atribuída a cada PHB estiver completa é necessário evitar que

o conjunto de admissões de um PHB interfira nas admissões de outro. As regras garantem que a alocação de recursos para uma determinada classe de fluxos seja realizada independente da outra, e que todas elas respeitem a mesma largura de banda. Essas regras garantem a integridade das alocações com a definição dos limites de cada PHB, com as restrições dos valores máximos permitidos para cada um deles. Uma vez que o fluxo é admitido são garantidos mais ou menos recursos dentro dos limites impostos para o PHB, sem que isso interfira nas alocações de outras classes. Na verdade é realizada uma distribuição controlada de recursos para várias redes lógicas em uma mesma rede física. A otimização do uso da banda ocorre de maneira controlada dentro dos limites atribuídos a cada PHB para evitar que um conjunto de admissões inviabilize ou penalize severamente as admissões de outras classes.

As regras são divididas em dois grupos: a primeira regra mantém o critério para a definição dos limites de largura de banda; as demais mantêm os critérios para a admissão dos fluxos na rede. Como consequência, $\sum_{i=1}^n \text{alocado}_{PHB\ i} \leq \text{bandwidth}$, ou seja, cada PHB admite uma determinada quantidade de fluxos com recursos mínimos garantidos para cada um desses fluxos. Quando a porção do enlace atribuída a cada PHB estiver completa é necessário evitar que o conjunto de admissões de um PHB interfira nas admissões de outro. As regras garantem que a alocação de recursos para uma determinada classe de fluxos seja realizada independente da outra, e que todas elas respeitem a mesma largura de banda.

Essas regras garantem a integridade das alocações com a definição dos limites de cada PHB, com as restrições dos valores máximos permitidos para cada um deles. Uma vez que o fluxo é admitido são garantidos mais ou menos recursos dentro dos limites impostos para o PHB, sem que isso interfira nas alocações de outras classes. Na verdade é realizada uma distribuição controlada de recursos para várias redes lógicas em uma mesma rede física. A otimização do uso da banda ocorre de maneira controlada dentro dos limites atribuídos a cada PHB para evitar que um conjunto de admissões inviabilize ou penalize severamente as admissões de outras classes.

A implementação das configurações com a ferramenta *tc* é estática e o BB mantém o controle das alocações com base nas heurísticas. A garantia mínima da alocação é mantida quando o $\min = \max$, ou seja, apenas um experimento pode alocar a banda mapeada para o PHB específico. A garantia mínima muda para mais ou para menos à medida que mais experimentos submetem fluxos que compartilham a largura de banda atribuída ao mesmo PHB.

5.4 Avaliação do Uso das Heurísticas

O uso das heurísticas propostas permite um controle mais eficiente da largura de banda do laboratório. O BB precisa manter o controle das alocações de banda para evitar que muitos experimentos sejam iniciados e sobrecarreguem a rede do laboratório. Nos cenários apresentados são submetidos diferentes agregados a essa rede, ou seja, um conjunto de pacotes IP com o cabeçalho marcado com o mesmo DSCP no campo *DS*.

Para a geração de fluxos e recuperação das estatísticas do tráfego submetido à rede foram utilizados os *softwares* RUDE (*Real-time UDP Data Emitter*) e CRUDE (*Collector for RUDE*) [76]. Em especial, o *software* RUDE foi produzido para suprir as deficiências de precisão do *software* MGEN [81]. O *software* RUDE utiliza um arquivo de *script* para realizar a emissão de tráfego na rede. O *script* informa parâmetros como o tempo de duração, início de cada fluxo, porta de origem, porta de destino, endereço de destino, tamanho dos pacotes, taxa de geração dos pacotes e, opcionalmente, o valor do campo *DS* do pacote. Já o programa CRUDE recebe o tráfego gerado pelo programa RUDE e cria um arquivo de *log*. A partir desse arquivo podem ser geradas informações para a análise do desempenho da rede, tais como: vazão, atraso fim-a-fim, perda de pacotes e *jitter*.

Os fluxos submetidos à arquitetura foram gerados com os *softwares* RUDE e CRUDE para simular, de forma sistemática, o comportamento de tráfegos de pacotes com diferentes marcações no campo *DS* e com valores específicos de vazão. Para recolher corretamente os valores da vazão e perda de pacotes criou-se uma nova versão do *software Qosplot* [77].

A modificação foi necessária devido à incoerência entre os valores de vazão indicados pelo *software* GKrellM [82] e os do *software Qosplot*. A incoerência ocorre quando este *software* incrementa o tamanho dos pacotes IP coletados, indicando uma vazão superior à descrita pelo GKrellM. Mas não há a necessidade de incrementar o tamanho do pacote porque o *software RUDE* já faz esse incremento na montagem do pacote UDP (*User Datagram Protocol*) submetido à rede. Em função disso, as seguintes linhas foram comentadas no arquivo *qosplot.c*:

```
stepQoSChars[streamIndex][xStepIndex].bytesReceived+=packet.size+8+20;
QoSChars[streamIndex].bytesReceived+=packet.size+8+20;
```

O *software Qosplot* é capaz de ler um arquivo de *log* em formato texto gerado pelo CRUDE e produzir uma saída de dados utilizada pelo *Gnuplot* para gerar os gráficos.

O experimento BB permite escolher quais *hosts* devem receber configurações *DiffServ* de acordo com as necessidades do experimento que irá utilizar essas configurações.

Para avaliar a submissão de fluxos no domínio, os *hosts* HELIOS e POSEIDON foram escolhidos como roteadores de borda. Nesses *hosts* foram inseridas disciplinas de fila que limitam a taxa de entrada de pacotes, assegurando que os fluxos que ultrapassarem a taxa pré-estabelecida serão descartados antes mesmo de entrarem na pilha de comunicação da interface de ingresso.

Nesses *hosts* realizou-se também o controle de tráfego com o policiamento, para determinar a taxa de entrada de pacotes no domínio. Dessa forma, os demais *hosts* do domínio, ou seja, os nós de núcleo, precisam apenas encaminhar os fluxos previamente filtrados. A Fig. 3.3 exemplifica a criação das disciplinas de fila, classes e filtros para comportamentos AF1, AF2, AF3, AF4, EF e BE. Para cada comportamento AF, três sub-classes são definidas. Como exemplo, para AF1 existem as sub-classes AF11, AF12 e AF13.

5.4.1 Teste #1: Vazão dos Fluxos Ultrapassa a Largura de Banda Global

Neste cenário, a largura de banda da rede é limitada em 10MBps (Megabytes/segundo) e não são utilizadas as heurísticas propostas. A vazão de cada agregado é limitada de acordo com a distribuição de banda descrita na Tab. 5.10. A vazão submetida por cada agregado é mostrada na Tab. 5.11. A Fig. 5.7 ilustra o resultado da submissão neste cenário.

PHB	Vazão Mínima Garantida (MBps)	Vazão Máxima Permitida (MBps)
AF11	1	5
AF21	1	4
AF22	1	3
BE	1	3
EF	1	3

Tab. 5.10: Distribuição de Largura de Banda para cada PHB.

A consequência de assegurar a vazão para cada agregado sem respeitar os limites do enlace é a diminuição da QoS para a aplicação que submete os fluxos. Na esperança de garantir uma vazão adequada para todos os agregados, estes competem entre si e todos são igualmente penalizados porque não há uma distribuição adequada da banda. Esse cenário é baseado em uma configuração *DiffServ*

PHB	Período (segundos)	Pacotes/segundo	Bytes/pacote
AF11/AF21	1-5	2000	500
AF11/AF21	5-10	2000	1000
AF11/AF21	10-15	2000	1500
AF11/AF21	15-20	2000	2500
AF11/AF21	20-25	2000	500
AF11/AF21	25-30	2000	1000
AF11/AF21	30-40	2000	1500
AF22	1-40	2000	1500
BE	1-40	2000	1500
EF	1-40	2000	1500

Tab. 5.11: Vazão Gerada por Cada Agregado ao Longo do Tempo.

que tenta distribuir todos os recursos disponíveis para cada agregado quando a banda não estiver completamente utilizada.

Ao submeter individualmente cada um dos cinco fluxos à rede é esperado que a perda de pacotes seja mínima e que a vazão seja a máxima possível. Seguindo esse princípio, todos os fluxos deveriam ser capazes de submeter seus fluxos à rede, sem qualquer restrição, à priori. Assim, os fluxos AF11 e AF21 variam de forma semelhante o seu comportamento ao longo do tempo até o instante de 15 segundos. Os fluxos AF22, EF e BE submetem a vazão constante de 3000KBps.

Este cenário contempla os resultados da vazão e da perda de pacotes dos fluxos descritos anteriormente, quando submetidos ao mesmo tempo. Qualquer um dos fluxos que submeta à rede uma vazão maior do que a definida no roteador de borda, ou maior do que a definida para a sua classe, terá os seus pacotes descartados.

O gráfico da vazão representa o comportamento agregado AF que define a maior prioridade para o fluxo AF11 e a menor para o fluxo BE. À medida que os fluxos AF11 e AF21 aumentam a sua vazão, os demais fluxos são penalizados. Até o instante de 15 segundos, AF22, BE e EF são penalizados. Isso ocorre porque, para a configuração *DiffServ* apresentada, AF11 e AF21 estão inseridos na disciplina GRED com uma largura de banda maior e alta prioridade de encaminhamento de pacotes em relação aos demais fluxos AF.

Na configuração, o fluxo do tipo EF foi mapeado para uma disciplina de fila FIFO com tamanho de fila reduzido, mas percebe-se que essa opção não é a ideal para concorrer com os outros fluxos AF em uma disciplina HTB. A preocupação nesse mapeamento EF foi apenas o de oferecer baixo *delay*, sem a preocupação com a preempção que seria, a priori, a mais desejada. O objetivo desse cenário é mostrar que as configurações de disciplinas de fila não privilegiam o descarte dos pacotes inadimplentes, mas sim a redistribuição deles. É possível outra configuração *DiffServ* que privilegie os fluxos EF mas, ainda assim, a excessiva competição entre os fluxos admitidos reduz a QoS do enlace, com a ocorrência de altos índices de perda. O cenário é baseado em uma configuração que tenta distribuir todos os recursos disponíveis para cada agregado quando o enlace não estiver completamente utilizado. Os fluxos BE e EF participam da distribuição de banda mas, por estarem em outras classes, a configuração apenas lhes assegura a vazão mínima garantida em situações de congestionamento.

De 15 a 20 segundos, AF11 submete à rede a vazão de 5MBps para a sua classe. Nesse período, AF21 também é penalizado, mas mantém a sua vazão acima da largura de banda mínima assegurada. Como a largura de banda máxima foi limitada, os fluxos utilizam praticamente toda a banda. No instante de 20 segundos, AF11 e AF21 reduzem a sua vazão, o que permite que os demais fluxos ocupem a banda restante.

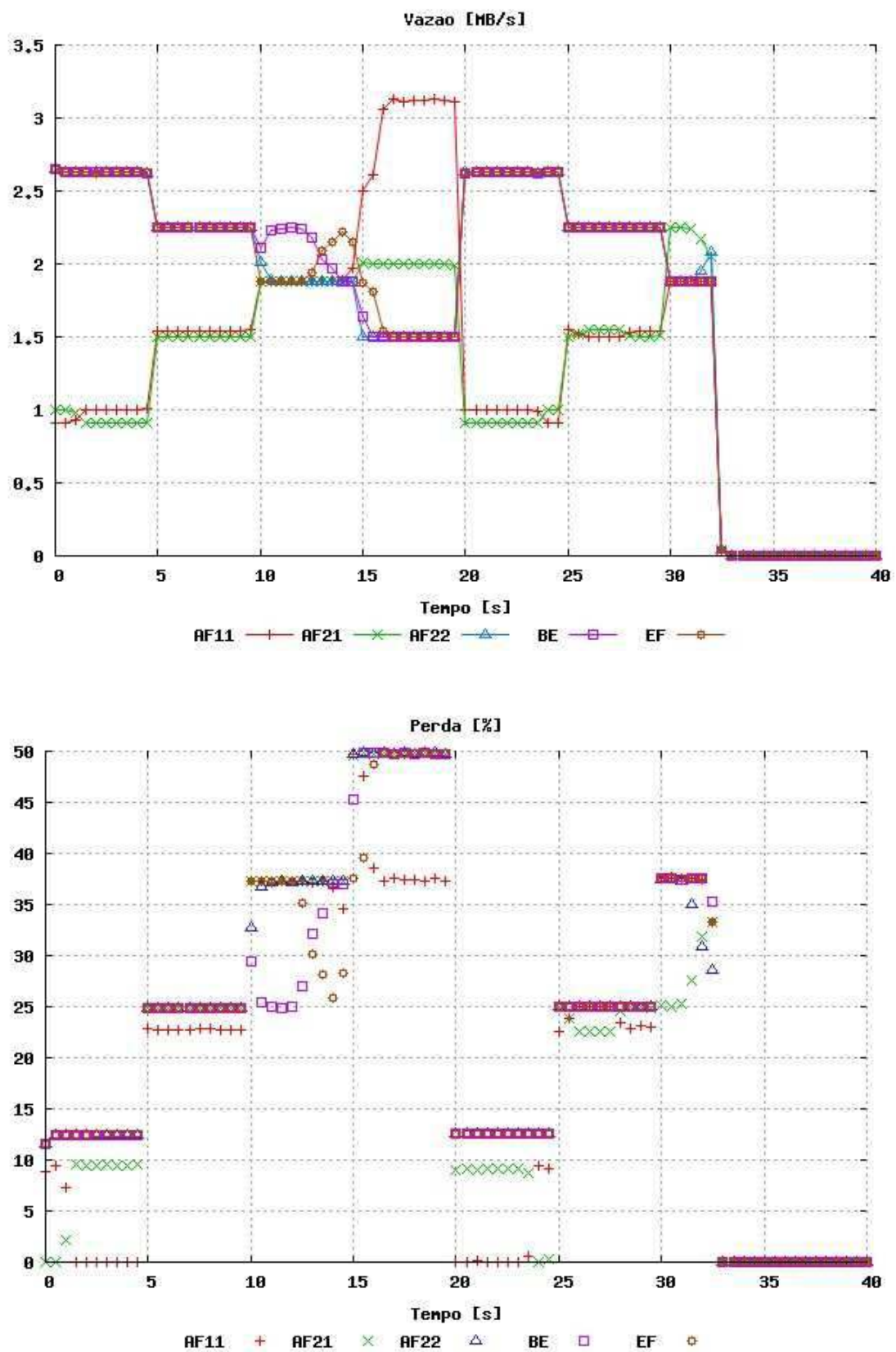


Fig. 5.7: Vazão e Perdas Obtidos com a Garantia de Vazão para Cada Agregado.

PHB	Vazão Mínima Garantida (MBps)	Vazão Máxima Permitida (MBps)
AF11	1	3
AF21	1	2
AF22	1	1
BE	1	1
EF	3	3

Tab. 5.12: Redistribuição de largura de banda para os agregados.

Na sequência e à medida que os fluxos AF11 e AF21 aumentam a sua vazão, os demais fluxos são novamente penalizados. Percebe-se que a preempção de recursos ocorre entre os fluxos das classes AF configurados com a disciplina GRED, mas os serviços BE e EF são tratados igualmente e esse comportamento é indesejável. Esse problema ocorre porque é permitida a entrada no domínio de fluxos de até 10MBps. A configuração GRED para as classes AF permite distribuir adequadamente os recursos entre AF11, AF21 e AF22, mas a alta vazão de entrada permitida para BE penaliza o comportamento do agregado EF, ou seja, é necessário um controle mais preciso dos fluxos na entrada do domínio para auxiliar as disciplinas de fila a darem prioridade no encaminhamento dos agregados. Da mesma forma, é indesejável que agregados de alta prioridade tenham altos índices de perda.

A arquitetura *DiffServ* procura promover a reserva de parâmetros de QoS a maior parte do tempo, mas a dinâmica da rede, limitação e carga de banda, processamento de pacotes pela interface de rede, limitação física do meio, por exemplo, impedem a garantia absoluta de tais parâmetros. Como exemplo, após os 30 segundos iniciais, a consulta pela atividade da sessão remota estabelecida entre os *host* de origem e destino, através do *hub Ethernet*, faz a aplicação de geração de fluxos permanecer inativa, refletindo a queda abrupta nos valores dos gráficos.

5.4.2 Teste #2: Vazão dos Fluxos Compatível com a Largura de Banda Global

Para resolver os problemas do cenário anterior, foi proposto o aumento da largura de banda sem a utilização das heurísticas propostas. Neste cenário, a largura de banda foi incrementada de 10MBps para 20MBps. A mesma vazão dos agregados foi novamente submetida. A Fig. 5.8 ilustra o resultado da submissão neste cenário. A consequência direta do aumento da largura de banda é o aumento da QoS de vazão oferecida para as aplicações. No entanto, o custo associado ao aumento da oferta de recursos pode não ser proporcional ao seu uso, resultando em desperdício de utilização. Neste cenário, os fluxos BE de menor prioridade obtiveram a mesma vazão que os fluxos EF de maior prioridade e esse comportamento é indesejável.

5.4.3 Teste #3: Admissão de Fluxos Respeitando as Heurísticas

O cenário 3 tem o objetivo de se aproximar do cenário anterior com uma largura de banda de 10MBps. Para isso, é considerado que a distribuição adequada dos recursos da rede é preferível à competição natural entre os agregados e que deve ser mantida a reserva de recursos para os agregados EF em detrimento dos demais. Os agregados AF devem manter a competição por recursos entre si, mas para BE é preferível uma acentuada penalização em benefício dos demais.

Esse cenário é possível com o uso das heurísticas propostas nos filtros de ingresso. A mesma vazão dos cenários anteriores é submetida por cada agregado. No entanto, é promovida uma realocação de banda com o experimento BB, como mostra a Tab. 5.12, para respeitar as heurísticas. Essa realocação é realizada pelo administrador do WebLab para que a distribuição de recursos respeite as necessidades das aplicações. A Fig. 5.9 ilustra o resultado da submissão neste cenário.

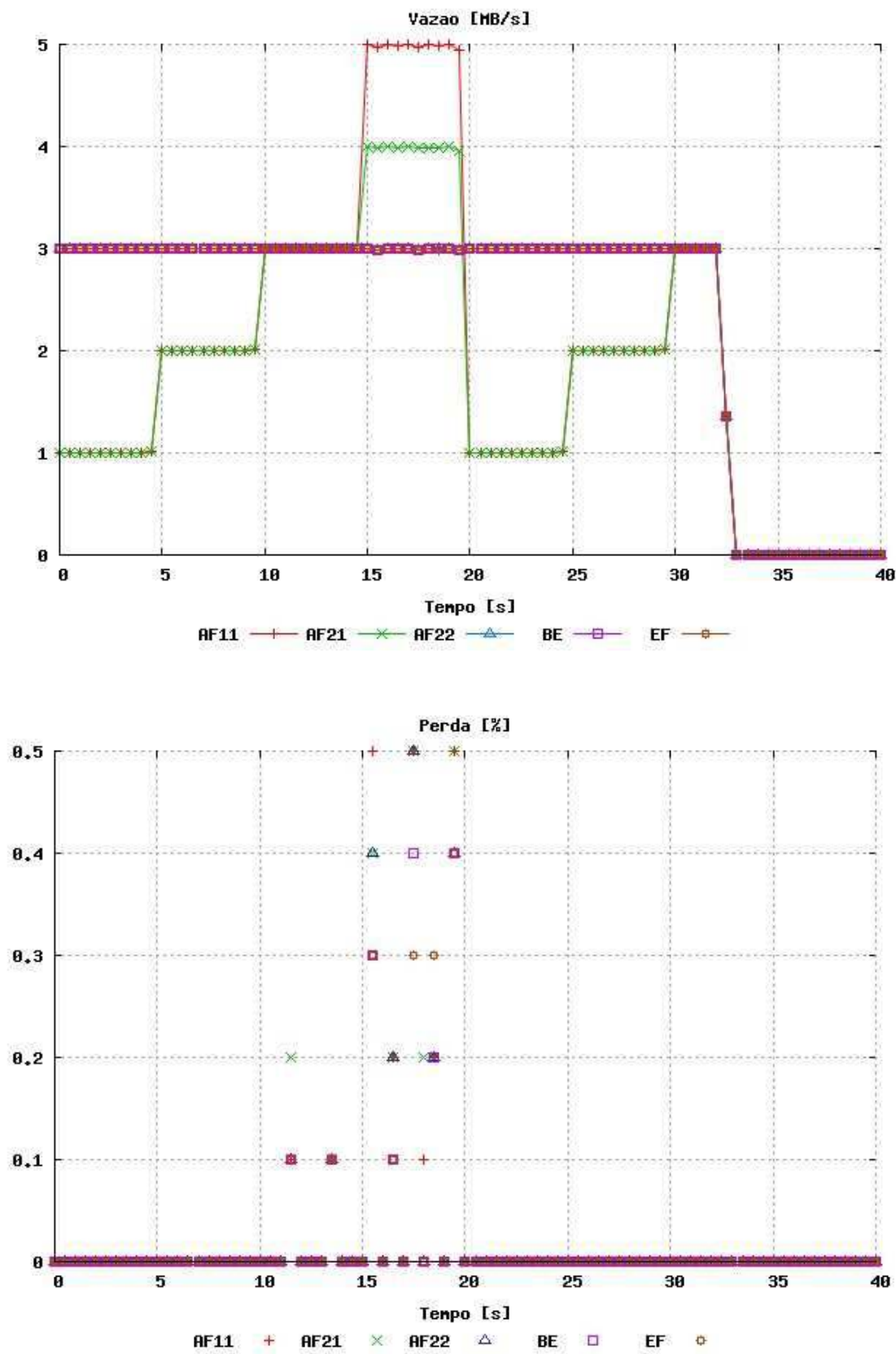


Fig. 5.8: Vazão e Perdas Obtidos com o Aumento da Largura de Banda.

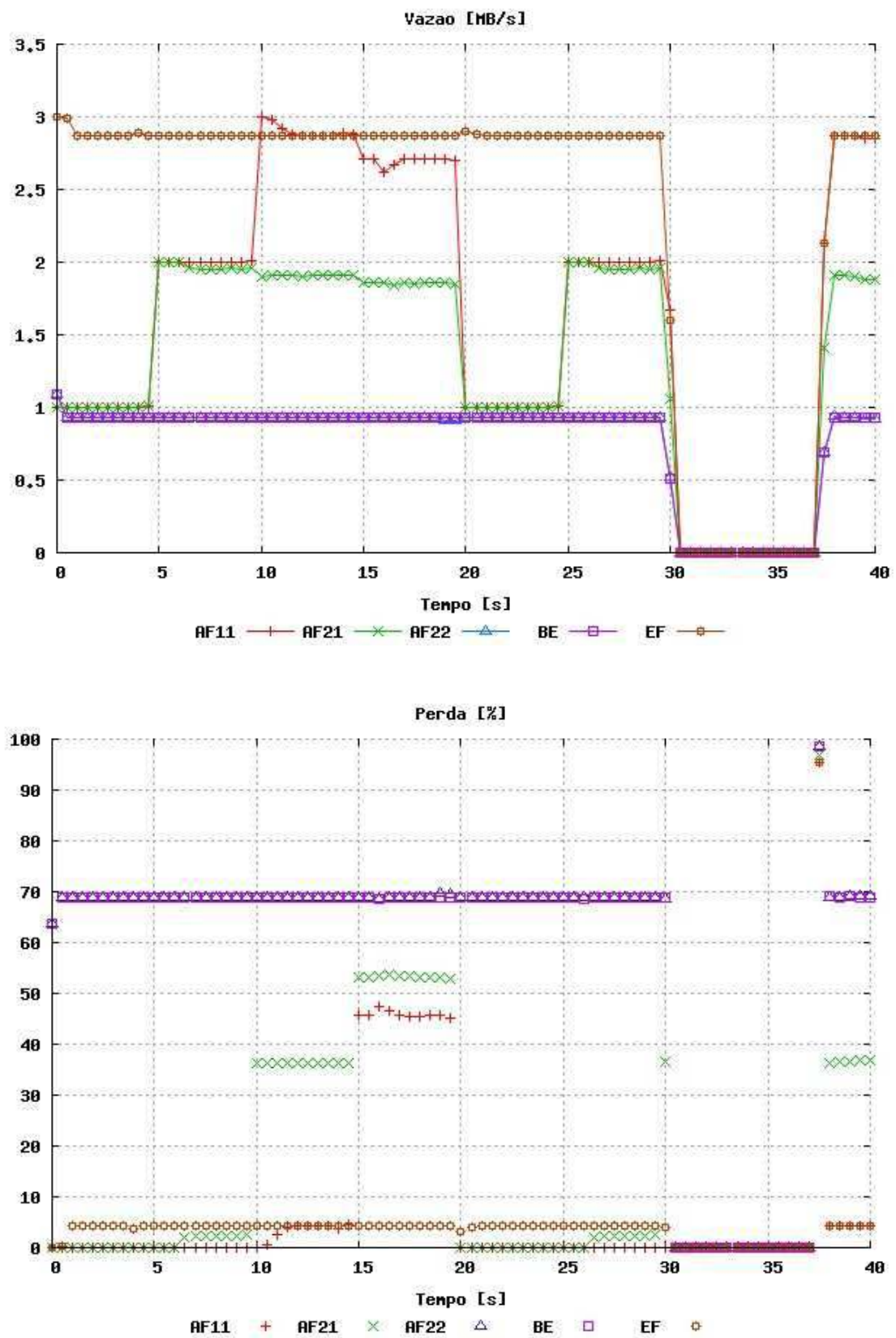


Fig. 5.9: Vazão e Perdas Obtidos com a Atribuição de Heurísticas.

Como consequência direta dessa redistribuição tem-se uma redução significativa da perda de pacotes com a garantia da vazão para os agregados restrita nas porções alocadas de cada PHB. Outra consequência é que se tem uma perda acentuada de pacotes para os agregados BE de menor prioridade, mas essa perda é controlada e tem o intuito de beneficiar os agregados EF de maior prioridade de encaminhamento. A redistribuição de recursos reduz significativamente a concorrência entre os agregados associados a PHBs diferentes. Um resultado importante é que houve a garantia de submissão do agregado EF com perdas reduzidas. Os agregados ainda trafegam com a mesma vazão até atingirem o roteador de ingresso no domínio. Após, cada fluxo será filtrado com base nas heurísticas implementadas nos filtros de ingresso. O trecho a seguir mostra o policiamento dos agregados. A configuração completa é descrita no Apêndice A.

```
#--Filtro para agregado AF11
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip tos 0x28 0xff police rate 3mbps burst 80k drop flowid 1:0
#--Filtro para agregado AF21
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip tos 0x48 0xff police rate 2mbps burst 80k drop flowid 1:0
#--Filtro para agregado AF22
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip tos 0x50 0xff police rate 1mbps burst 80k drop flowid 1:0
...
#--Filtro para agregado BE
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip tos 0x0 0xff police rate 1mbps burst 80k drop flowid 1:0
#--Filtro para agregado EF
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip tos 0xb8 0xff police rate 3mbps burst 80k drop flowid 1:0
```

5.4.4 Teste #4: Controle da Admissão de Fluxos Respeitando as Heurísticas

Este cenário tem o objetivo de justificar o uso do BB para a admissão de agregados nos experimentos. A largura de banda foi limitada em 10MBps com a presença de heurísticas implementadas nos filtros de ingresso. A mesma vazão dos agregados é submetida à rede, mas com dois fluxos EF submetidos ao mesmo tempo. A Fig. 5.10 ilustra o resultado da submissão neste cenário.

O resultado obtido mostra que a configuração *DiffServ* para EF não tem prioridade de preempção de banda, mas sim prioridade de encaminhamento de pacotes. Essa prioridade é oferecida pela disciplina de fila implementada na classe (PFIFO com tamanho de fila reduzido, garantindo alta vazão, baixo *jitter* e perda reduzida). Ambos os fluxos EF dividem as perdas, mas não as distribuem para os outros PHBs. Por isso, existe a necessidade do controle de admissão para impedir que as aplicações submetam mais agregados do que a porção reservada para o PHB suporta. A implementação das heurísticas nos filtros de ingresso permitem que os dois fluxos sejam admitidos, mas o BB precisa implementar as heurísticas de controle de recursos de cada porção de recursos de banda mapeados para cada PHB com a finalidade de evitar altas perdas resultantes da distribuição inadequada de recursos entre os experimentos realizados simultaneamente.

5.5 Experimento Gerador de Fluxos

O experimento Gerador de Fluxos contempla parte das características de sistemas de tempo-real que têm de processar muitas atividades ao mesmo tempo dentro de um espaço de tempo preciso. Portanto, eles são melhor reconhecidos por suas propriedades de limite de tempo e concorrência [83].

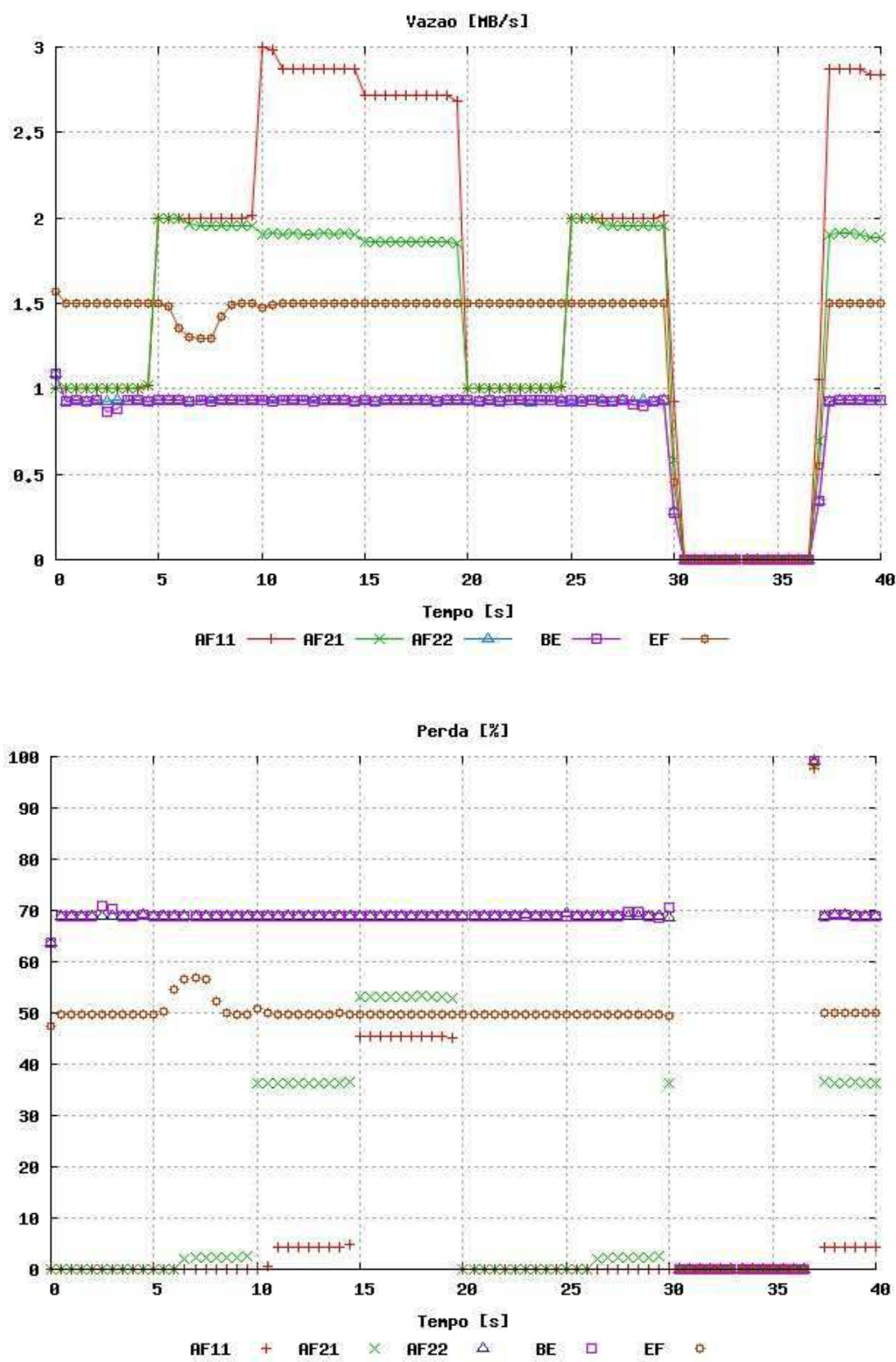


Fig. 5.10: Vazão e Perdas Obtidos com Dois Agregados EF Concorrentes.

A arquitetura do experimento Gerador de Fluxos é ilustrada na Fig. 5.11. Esse experimento é uma proposta que, juntamente com o experimento *Bandwidth Broker*, valida o uso de experimentos *DiffServ* no domínio do WebLab. A Fig. 5.12 mostra a seqüência de execução do experimento. O processo tem início com a submissão de fluxos pelo objeto GeradorFluxos que envia por SOAP a lista com o início da submissão, quantidade de pacotes por segundo e quantidade de bytes por pacote de cada período do tráfego. O *objeto servidor servicoRude* recebe a lista com o comportamento do tráfego, ativa o *servicoCrude* no *host* de destino do fluxo e inicia a geração do tráfego. O *objeto servidor servicoCrude* coleta o tráfego gerado. Nesse ínterim, o *objeto GeradorFluxos* consulta periodicamente a pasta compartilhada do experimento para verificar o fim do processo de submissão e de coleta do tráfego. Ao receber a sinalização na pasta compartilhada sobre o fim do processo de coleta

de dados, o *objeto GeradorFluxos* solicita a ativação da plotagem dos dados. Novamente, o *objeto GeradorFluxos* realiza consultas periódicas na pasta compartilhada do experimento à espera do fim da plotagem. O *objeto servicoCrude* sinaliza o *objeto GeradorFluxos* gravando a informação do fim da plotagem na pasta compartilhada. Ao receber essa confirmação, o *objeto GeradorFluxos* exibe o resultado gráfico da submissão do tráfego na rede interna do WebLab.

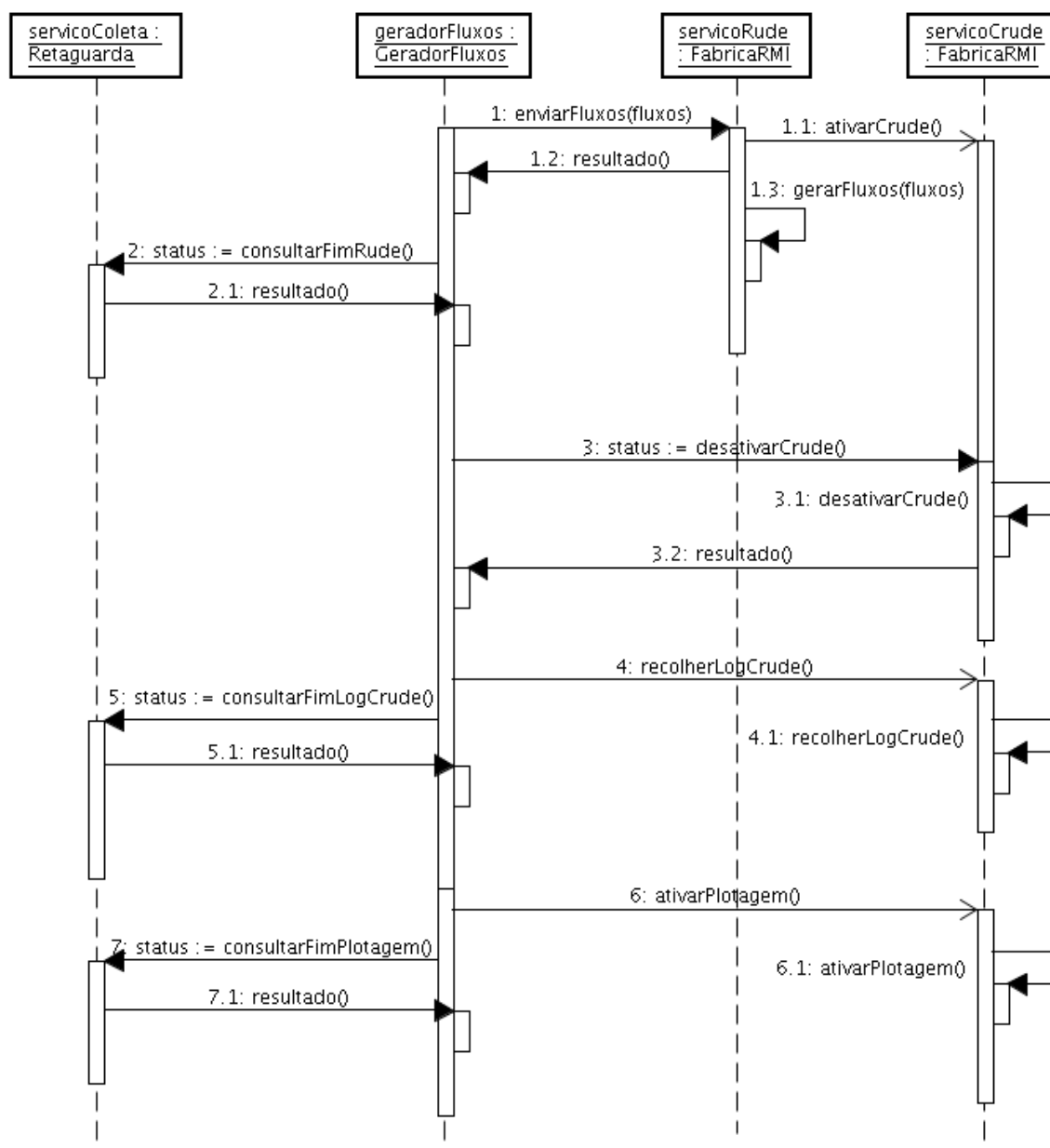


Fig. 5.12: Diagrama de Seqüência do Experimento Gerador de Fluxos.

Esse experimento submete remotamente aos *hosts* do laboratório fluxos UDP. Para os estudos de caso apresentados foram escolhidos os *hosts* HELIOS como o *host* de origem e o *host* POSEIDON como o *host* de destino. Os fluxos são gerados com os *softwares* RUDE, capturados pelo *software* CRUDE e plotados com os *softwares* Qosplot e Gnuplot.

O experimento *Bandwidth Broker* prepara a configuração *DiffServ* do domínio para cada usuário do laboratório. Dessa forma, o usuário pode simular a passagem de dados com e sem a presença das configurações *DiffServ* disponíveis para ele. O processamento das plotagens não é imediato e, por

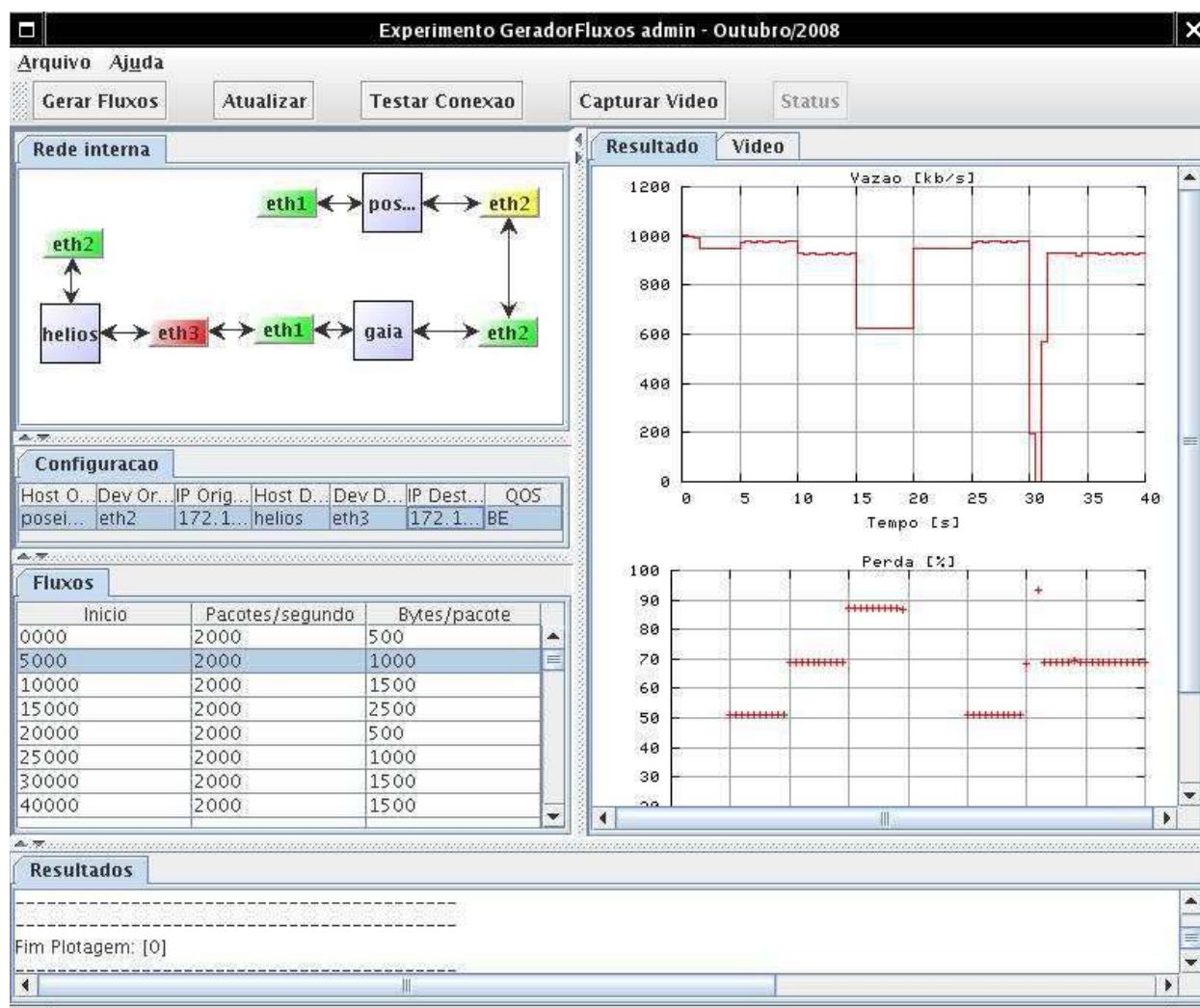


Fig. 5.13: Experimento Gerador de Fluxos.

isso, o experimento precisa realizar consultas periódicas sobre o *status* dos processos de submissão, coleta de dados dos fluxos e finalização da plotagem. Esse processamento deve ser realizado no intervalo de tempo válido definido para cada um dos processos.

A Fig. 5.13 ilustra o experimento Gerador de Fluxos. O usuário pode selecionar as interfaces de rede de origem e destino do fluxo e testar a conexão com o uso de um serviço de interação que executa remotamente o comando *ping*. Para testar a configuração *DiffServ* preparada como experimento BB, o usuário seleciona a interface de origem e destino do fluxo, escolhe o tipo de serviço de encaminhamento do fluxo e cria um fluxo. O fluxo é criado informando o início, pacotes/segundo e bytes/pacote de cada período da submissão do tráfego. Ao clicar no botão “Gerar Fluxos” o usuário envia as informações dos *hosts* de origem e destino, os fluxos e a qualidade do serviço para a submissão de tráfego. A Fig. 5.13 mostra o resultado da submissão da mesma vazão da Tab. 5.11, com o serviço de encaminhamento do tipo BE. Os resultados da vazão e perda validam a aplicação das heurísticas como alternativa de distribuição adequada da largura de banda para os experimentos. As perdas que ocorrem quando a vazão ultrapassa 1Mbps é percebida, empiricamente, como uma conexão mais lenta para a transferência de pacotes IP.

Com a alteração do serviço de encaminhamento para AF11 uma vazão de até 3Mbps é permitida na submissão, com perdas reduzidas, como mostra a Fig. 5.14. A Tab. 5.13 mostra outro exemplo de tráfego submetido à rede no experimento Gerador de Fluxos.

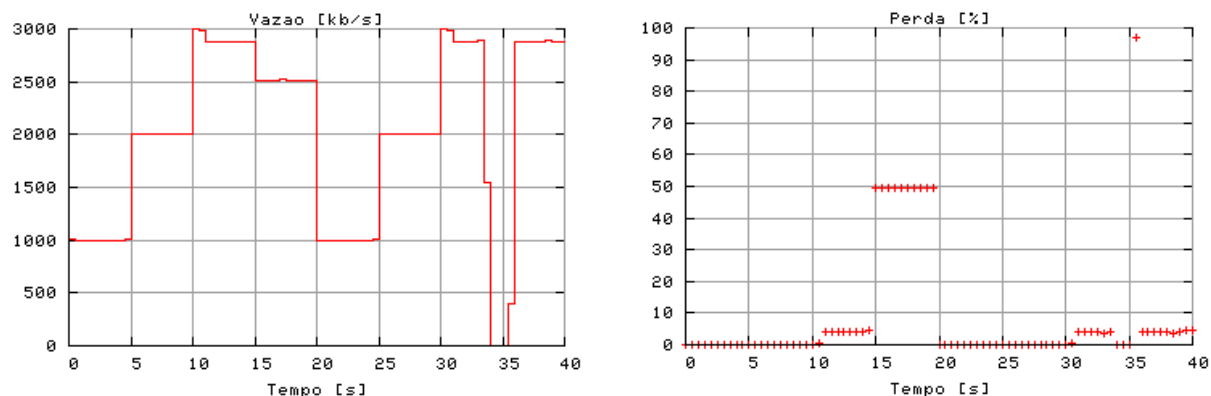


Fig. 5.14: Vazão e Perdas para a Submissão do Tipo AF11.

Tempo (ms)	Pacotes/Segundo	Bytes/Pacote
0	20000	250
5000	40000	500
12500	13500	500
16000	4000	100
22000	5000	4380
27500	8000	1500
30000	8000	1500

Tab. 5.13: Fluxos gerados pela aplicação Gerador de Fluxos.

5.5.1 Teste #1: Análise da Vazão em Fluxos Simulados

Os experimentos de submissão de fluxos simulados são úteis para o estudante de redes de computadores porque permitem analisar o comportamento de diferentes submissões em um domínio controlado remotamente. Diversos tipos de análises são possíveis, como será apresentado a seguir. O resultado da submissão dos valores da Tab. 5.13 é exibido para o usuário como mostra a Fig. 5.15.

A comparação entre o gráfico da plotagem dos valores da vazão e da perda de pacotes permite observar que, no intervalo de 5 a 12.5 segundos a submissão de uma grande quantidade de pacotes não é suportada pelo *switch gigabit* do laboratório e gera uma alta perda de pacotes. O intervalo entre 22 e 27.5 segundos demonstra que deve existir um equilíbrio entre o número de pacotes

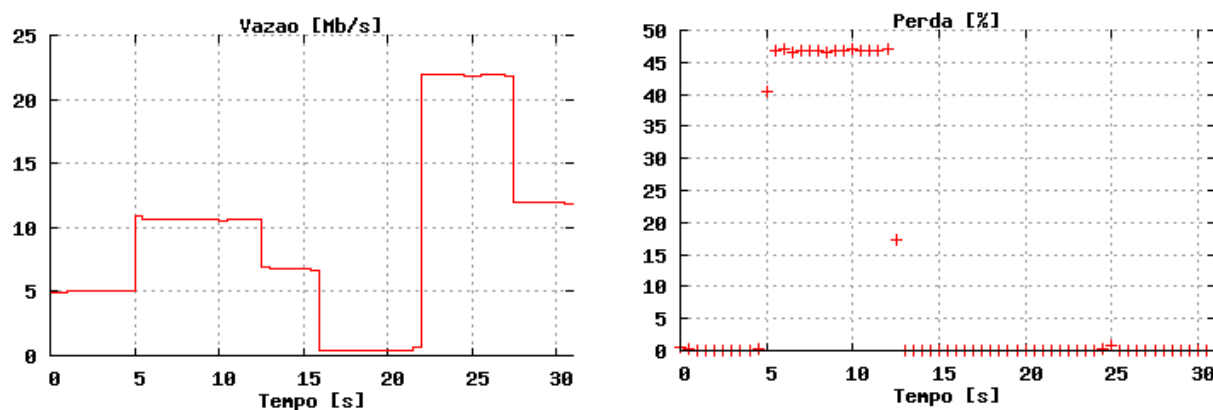


Fig. 5.15: Vazão e Perda sem Diferenciação de Serviços.

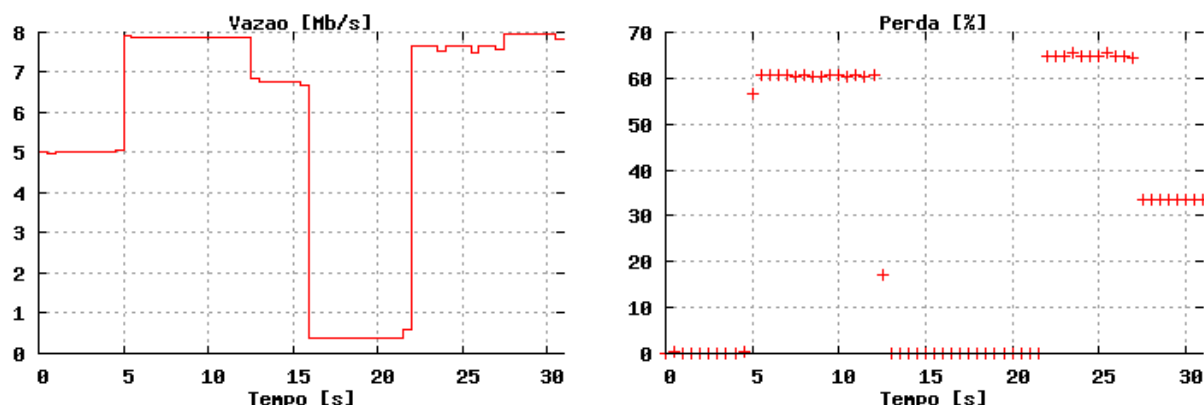


Fig. 5.16: Vazão e Perda com Diferenciação de Serviços.

enviados e o tamanho em bytes de cada um deles para gerar uma alta vazão (acima de 20MBytes/s) sem perda substancial de pacotes. Como exemplo, o cálculo da vazão é realizado da seguinte forma: $250 \text{ Bytes/pacote} \text{ com } 200 \text{ pacotes/s} = 50000 \text{ Bytes/s}$. Assumindo uma estimativa de $1 \text{ KB} = 1000 \text{ B}$, tem-se uma vazão de 50 KB/s . Para os mesmos valores da Tab. 5.13 foi aplicada a configuração *DiffServ* promovida pelo experimento *Bandwidth Broker*.

A Fig. 5.16 ilustra este comportamento. Foi aplicada uma limitação da largura de banda de 8MBytes com o PHB AF11. Qualquer um dos fluxos que submeta à rede uma vazão maior do que a definida por essa classe, terá os seus pacotes descartados.

5.5.2 Teste #2: Análise da Vazão de Fluxos de Vídeo

Os experimentos de tempo-real com submissão de fluxos de vídeo são úteis para estudantes de redes de computadores porque permitem verificar de que forma os pacotes de dados UDP se comportam com a atribuição de QoS. Este experimento descreve o comportamento de um fluxo de vídeo sob monitoramento face ao contrato estabelecido. O não cumprimento do mesmo pode implicar na reclassificação do fluxo segundo a política previamente acordada [84] [85]. Escolheu-se previamente monitorar apenas o fluxo AF11, informando essa decisão como parâmetro para o *objeto servidor Multimídia*. A Fig. 5.17 ilustra este experimento. A *webcam* é um dos recursos do *host* HELIOS, *host* de origem dos fluxos. Ela possui as seguintes especificações: 320K pixels VGA, 1/4,5 Sensory inch CMOS, resolução VGA de até 640x480, 30fps (CIF). Para a captura dos fluxos foi utilizada a biblioteca *open source* JChart2D. O modelo da *webcam* permite que sejam enviados, em média, 30KB/s de dados de vídeo, com picos entre 15KB/s e 44KB/s. A Fig. 5.17 mais à esquerda exibe o monitoramento dos dados monitorados que foram obtidos sem quaisquer políticas de QoS relacionadas à vazão.

A Fig. 5.17 mais à direita reflete a vazão do fluxo de vídeo com a presença do monitoramento de QoS junto ao experimento de Geração de Fluxos. O monitoramento definiu uma política de que o fluxo deve sofrer uma atenuação para 2KB/s quando exceder os 30KB/s. A atenuação irá ocorrer por 5 segundos e a vazão pode alcançar até 5KB/s. Isso permite demonstrar que o comportamento de uma aplicação de fluxo não-simulado é similar ao comportamento dos fluxos dos experimentos anteriores.

Além disso, a disciplina de fila *HTB* criada para a classe AF11 do fluxo de vídeo auxilia no controle da largura de banda fora do limite em um dado enlace, permitindo compartilhar um enlace físico para simular enlaces mais lentos e enviar diferentes tipos de tráfego por diferentes enlaces simulados [86]. Adicionalmente possui os parâmetros *burst* e *cburst* que podem ser vistos como dois baldes de *tokens*, um para o valor da taxa mínima de envio (*rate*), e o outro para a taxa limite (*ceil*).

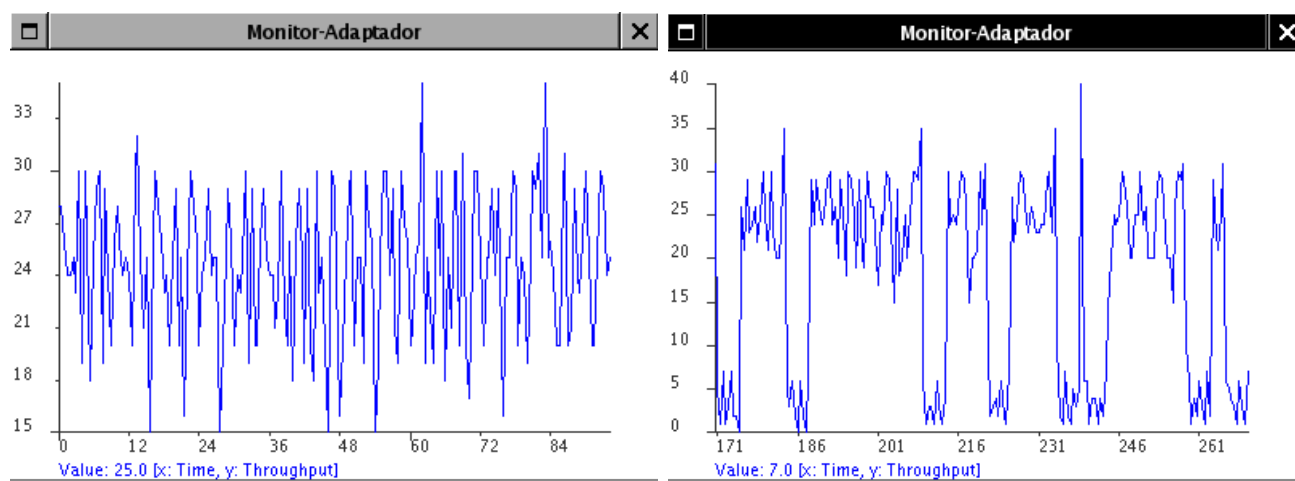


Fig. 5.17: Vazão do Fluxo de Vídeo com Ausência e Presença de Monitoramento.

Os parâmetros *burst* e *cburst* controlam a quantidade de dados que podem ser enviados na máxima velocidade da interface de rede sem atender outra classe. Quando o valor do parâmetro *burst* se torna negativo significa que o limite da taxa mínima foi superado, o mesmo ocorrendo com o parâmetro *cburst*. De posse dessas variáveis, o objeto servidor no *host* é capaz de verificar se os fluxos estão respeitando o contrato previamente estipulado de utilização da rede.

5.6 Negociação de Recursos Intra-domínio

Ao iniciar um experimento *DiffServ* a preparação do domínio é realizada de forma estática de acordo com a configuração específica do experimento. Dois ou mais experimentos Gerador de Fluxos podem ser acessados ao mesmo tempo. No entanto, a alocação de recursos é dinâmica, ou seja, um experimento apenas poderá ser iniciado caso haja recursos suficientes para que ele submeta os seus fluxos adequadamente.

As informações sobre quais fluxos e quais valores devem ser alocados são mantidos pelo BB. Logo, no início de cada experimento Gerador de Fluxos, este deve comunicar-se com o BB e solicitar a alocação de recursos com base na configuração atual. Caso seja possível a alocação, o BB armazena essa informação na base de dados, reduz a porção de banda disponível e permite o início do experimento.

É interessante notar que o experimento Gerador de Fluxos pode submeter fluxos além do permitido para o seu PHB. Nesse caso, a própria configuração irá impedir que os agregados excedentes passem pelo domínio. A configuração *DiffServ* mantida pelas heurísticas do BB garante que dois ou mais experimentos submetam fluxos à rede dentro dos limites do PHB.

Dessa forma, o BB é capaz de oferecer a garantia de submissão com QoS para os fluxos do experimento, evitando que muitos experimentos sejam iniciados e sobrecarreguem a rede. Um problema potencial é que a rede pode ser subutilizada caso nenhum fluxo seja submetido por um experimento, evitando que outros experimentos sejam iniciados. No entanto, uma vez que o experimento seja iniciado, existe a garantia de que a QoS oferecida não irá sofrer perda de desempenho em virtude do aumento ou diminuição da quantidade de experimentos. Essa característica é importante para o experimento Gerador de Fluxos porque assegura a coerência dos resultados. Uma alternativa para permitir que mais usuários utilizem a rede do WebLab é distribuir corretamente as porções da banda entre os experimentos. Isso permite que muitos experimentos sejam iniciados, mas com a garantia de vazão mínima dependente do número de experimentos admitidos ao mesmo tempo. O BB mantém,

de acordo com as alocações dinâmicas, a quantidade ideal de experimentos que podem ser iniciados sem sobrecarregar a rede. A Tab. 5.14 mostra um exemplo de SLA para dois experimentos agendados para o mesmo período.

IDSLA	IDUsuario	IDExp	QOS
1	1	A	AF11,0x28,1,MBps,3,MBps;
2	1	B	AF11,0x28,1,MBps,3,MBps;

Tab. 5.14: SLA dos experimentos A e B.

A Tab. 5.15 ilustra um exemplo de negociação de recursos entre dois experimentos. O significado das colunas é descrito a seguir:

- Op: Índice da operação do experimento para a solicitação de alocação de largura de banda do PHB;
- ID Exp: Índice do experimento;
- Solic: Quantidade de largura de banda solicitada para alocação;
- R2: Regra da heurística 2;
- iQoS: Elemento do BB que promove a realocação de largura de banda;
- Alocado: Valor mantido no BB (Tabela PHB) para indicar a alocação global e atual de recursos;
- Ativo: Valor mantido no BB (Tabela PHB) para indicar a quantidade de operações de alocação ativas;
- UsoReal: Valor máximo de largura de banda permitido para o PHB / quantidade de operações de alocação ativas. Esse valor é utilizado para consultar a distribuição real de recursos para cada agregado no experimento.
- Aceito: Indica se a operação de solicitação de recursos pelo experimento foi ou não aceita;
- Alloc Exp: Indica a quantidade de recursos alocados para o agregado no experimento (Tabela PHB_EXPERIMENTOS).

A quantidade de experimentos que podem utilizar a porção de banda do PHB com a garantia mínima de vazão é determinada dinamicamente de acordo com a alocação no BB e os valores de alocação mínimo e máximo permitidos para o PHB. Cada solicitação apresentada na Tab. 5.15 é feita para o PHB AF11 que reserva recursos de banda para uma vazão de até 3MBps com o uso das heurísticas.

Na operação 1, o experimento B solicita a alocação de 1MBps da largura de banda disponível de AF11. A regra da heurística 2 verifica que o valor solicitado é menor do que o máximo permitido para o PHB e que, com base na alocação atual, mais banda pode ser alocada para o experimento. Então, o BB armazena as informações sobre a alocação, a quantidade de operações de alocação ativas e o uso real da banda. Para apenas um fluxo toda a banda pode ser utilizada. A operação de alocação é aceita e o BB armazena a quantidade de banda alocada para o PHB no experimento.

Na operação 2 o mesmo procedimento anterior é adotado, mas agora os dois experimentos irão dividir a largura de banda disponível.

			R2	iQoS	Alocado	Ativo	UsoReal	Aceito	Aloc Exp
Op	ID Exp	Solic	-	-	0	0	0	-	0
1	B	1	Sim	-	1	1	3	Sim	1
2	A	2	Sim	-	3	2	1.5	Sim	2
3	A	1	Não	-	-	-	-	Não	-
4	B	-1	-	-	2	1	3	-	0
5	A	1	Sim	-	3	1	3	Sim	3
6	A	-3	-	-	0	0	0	-	0
7	B	3	Sim	-	3	1	3	Sim	3
8	B	-3	-	-	0	0	0	-	0
9	B	1.5	Sim	-	1.5	1	3	Sim	1.5
10	A	3	Não	Sim	2.5	2	1.5	Sim	1
11	A	3	Não	Não	-	-	-	Não	-

Tab. 5.15: Exemplo de negociação de recursos intra-domínio.

Na operação 3 o experimento A não consegue alocar mais recursos porque a regra da heurística 2 não é satisfeita. As informações sobre os valores alocados anteriormente são mantidas, mas a alocação é recusada.

Na operação 4 o experimento B é encerrado e é liberado 1MBps de banda alocada.

Na operação 5 o experimento A repete a solicitação de alocação da operação 3 e consegue adquirir a banda solicitada. O BB registra mais 1MB alocado para o experimento. As operações de 6, 7, 8, 9 e 11 se comportam de forma semelhante às operações apresentadas anteriormente.

Na operação 10 a regra da heurística 2 não é satisfeita, mas ainda existem recursos disponíveis para a alocação. Nesse caso, o método iQoS atua e refaz a solicitação. O algoritmo verifica se a largura de banda disponível garante a quantidade de banda mínima que o experimento deve possuir para funcionar adequadamente. Caso essa premissa seja satisfeita, a banda mínima é alocada, mas o usuário pode ou não aceitar essa alocação. O algoritmo iQoS é um método implementado no BB que garante a realocação mínima de recursos quando a solicitação inicial não puder ser atendida. O trecho a seguir ilustra o funcionamento do algoritmo de negociação de recursos.

```

negociador_intra_dominio(){

    //Heurística 2
    //O valor solicitado é menor do que o máximo?
    if ( max_PHB - solicitado_PHB >= 0 )
        //Heurística 2
        //Com base no que está alocado, é possível alocar mais?
        if ( alocado_PHB + solicitado_PHB <= max_PHB )
            aprovado = true;
        else
            aprovado = iqos( solicitado_PHB );

    if (aprovado){
        //Atualiza a alocação do PHB no experimento
        alocado_PHB_Exp += solicitado_PHB;

        //Atualiza a alocação do PHB no BB
    }
}

```

```
    alocado_PHB_BB += solicitado_PHB;
    ativo_PHB_BB++;

    //Heurística 3: garantia mínima de recursos
    usoReal_PHB_BB = max_PHB / ativo_PHB_BB;
} //fim if
} //fim negociador_intra_dominio

boolean iqos ( solicitado_PHB ) {

    boolean aprovado = false;
    //Existem recursos mínimos?
    if ( max_PHB - alocado_PHB >= min_PHB ) {
        solicitado_PHB = min_PHB;
        aprovado = true;
    } //fim if
    return aprovado;
} //fim iqos
```

5.7 Criação de Novos Experimentos

A metodologia adotada para o desenvolvimento de experimentos tem o intuito de reduzir a necessidade de codificação. A estrutura básica baseia-se no modelo distribuído em três blocos da Fig. 4.5. Cada um desses blocos realiza uma tarefa bem definida no seu meio de atuação.

A escolha da arquitetura SOA contribuiu para simplificar a estrutura de comunicação agrupando as funcionalidades de um experimento em serviços. O uso de *Web Services* como blocos de construção desses serviços permite a criação de novos experimentos com a composição de serviços.

No entanto, o tempo de resposta empírico obtido com as implementações do protocolo SOAP estudadas (Axis1.4/Java e Axis2/Java) revelaram que o processamento de operações pelos *Web Services* representa um gargalo na *performance* da interação com os dispositivos remotos gerenciáveis. O envio de mensagens XML precisou ser condicionado para otimizar o tráfego de informações no servidor do laboratório. Esse processo envolveu a redução das mensagens XML utilizando parte da abordagem REST [51].

Em sistemas RESTful a informação necessária para acessar um recurso é informada na própria URL. Por isso, caso seja necessário enviar parâmetros, é indesejável que a URL seja muito extensa. Isso pode ser resolvido com o envio de uma mensagem XML com parâmetros que redirecionem para um recurso acessível em outra URL. Portanto, não são necessários o envelope SOAP, o corpo SOAP e nem o cabeçalho nas mensagens XML REST. A proposta desse trabalho utiliza parte da proposta REST ao manter cada *Web Service* associado a um recurso lógico específico (WSRecursos, WSEnlace, WSSessao, WSFabrica, WSBB, WSGeradorFluxos, entre outros) em uma URL específica. Complementar a isso, as mensagens SOAP são reduzidas e apenas o *payload* é enviado, mas os parâmetros para acessar os sub-recursos ainda permanecem encapsulados nas mensagens. Também é realizada a compressão dessas mensagens trafegadas entre a aplicação cliente e o *Web Service* do WebLab, como descrito no Apêndice C.

Como consequência, os *Web Services* foram utilizados apenas para encaminhar o tráfego de informações dos serviços de interação. Os objetos servidores associados aos *Web Services* efetivamente realizam o processamento nos *hosts* da rede interna do laboratório, reduzindo a carga de processamento do *host* servidor do WebLab. Essa solução está refletida nas arquiteturas do WebLab. O bloco cliente é responsável por organizar a lógica de controle da interação com os *Web Services*. A criação de um novo experimento envolve os seguintes passos:

1. bloco cliente: criação do método de chamada do *Web Service*;
2. bloco de comunicação: criação do *Web Service* do serviço;
3. bloco servidor: criação do objeto servidor que implementa a requisição do serviço do bloco cliente. O objeto deve ser adicionado à fábrica de objetos.
4. bloco servidor: criação da interface com a descrição dos métodos que podem ser acessados pelo *Web Service*;
5. bloco de comunicação: cópia da interface do objeto servidor;
6. bloco de comunicação: geração do *stub*;
7. bloco cliente: cópia do *stub* do *Web Service*.

Esse processo de desenvolvimento permite a criação de arquiteturas dinâmicas. Os novos experimentos acrescentam serviços à estrutura existente sem interferir no funcionamento dos outros serviços. A gerência administrativa dos serviços utilizados pelos experimentos organiza a instanciamento dos objetos servidores por meio das informações da base de dados. Dessa forma, apenas os objetos servidores necessários serão instanciados/removidos no início/fim de cada experimento.

5.8 Considerações Finais

O NetLab WebLab é capaz de manter diversos experimentos de redes com suporte a *DiffServ*, mas é necessário que exista uma adequada distribuição de recursos de largura de banda para a execução simultânea de diversos experimentos. O próximo capítulo faz as considerações finais do trabalho.

Capítulo 6

Conclusão e Trabalhos Futuros

Este capítulo apresenta a conclusão do trabalho com destaque para as contribuições e sugestões de trabalhos futuros.

6.1 Avaliação das Contribuições

A experimentação remota é de grande auxílio para o ensino de redes de computadores porque permite que um número maior de usuários tenha acesso a experimentos que exigem esforço de preparação, tanto do ambiente físico quanto do ambiente lógico. Nesse contexto, o NetLab WebLab cumpriu os seus objetivos ao propiciar um ambiente de testes didático para experimentos *DiffServ*.

Além disso, a arquitetura é genérica o suficiente para permitir que diversos experimentos de redes sejam adicionados sem interferir no funcionamento dos demais. As alterações na arquitetura do projeto GigaBOT simplificam o desenvolvimento dos experimentos, reduzem o espalhamento do código e oferecem uma solução para a coleta e representação virtual de recursos e sub-recursos. O padrão de projeto permite criar aplicações complexas e os experimentos *Bandwidth Broker* e Gerador de Fluxos são exemplos dessas aplicações. O primeiro oferece alternativas para simplificar o uso de configurações *DiffServ*, além de estender a arquitetura *DiffServ* para suprir parte de suas deficiências; o segundo valida a atribuição dessas configurações com testes dinâmicos submetidos ao domínio do WebLab.

Os ambientes de rede que fazem a distinção entre tráfegos e requisitos de QoS viabilizam o tráfego multimídia com alto desempenho. A distinção de tráfego traz a possibilidade da oferta de recursos de rede de acordo com o perfil dos usuários. Nesse sentido, os experimentos *DiffServ* contribuem para a análise do tráfego de informações em redes de alto desempenho. Um experimento como o Gerador de Fluxos pode ser visto como o estabelecimento de uma conexão com a Internet que, antes de iniciar a transmissão, exige que o BB faça a reserva de recursos e as negociações com base no SLA intra-domínio para que usuário seja capaz de escolher a QoS dos fluxos submetidos à rede.

Diversas arquiteturas podem ser formadas com a mesclagem dos 3 blocos de comunicação básicos. As novas arquiteturas para os experimentos são formadas com a interligação desses elementos básicos com a composição de serviços, novas interações nos elementos básicos e com as configurações da base de dados.

O projeto se preocupou com a interação do usuário e, por isso, foi proposto o uso de elementos visuais para representar os recursos físicos do laboratório. O uso de um *sandbox* JWS assinado digitalmente garante maior segurança na execução do experimento pelos participantes do laboratório. A infraestrutura de *software* do WebLab é baseada em *softwares* livres e de código-fonte aberto. Essa característica permite a distribuição de versões desse WebLab para serem utilizadas/modificadas

por outras instituições e/ou outras áreas de conhecimento. Como exemplo, uma dissertação de mestrado [23] apresentou experimentos para configuração de rotas dinâmicas seguindo parte da infraestrutura atual para o desenvolvimento de experimentos.

A segurança do acesso é mantida com os serviços de acesso que autenticam os participantes, e do ambiente de acesso aos experimentos. A finalização adequada garante o uso de recursos apenas durante o tempo da reserva do experimento. O uso da Computação Orientada a Serviços simplificou a criação de experimentos com o uso da composição de serviços, mas a modelagem da arquitetura para os experimentos também foi influenciada pela experiência adquirida com o uso do *software* de integração das aplicações cliente/servidor. A experiência adquirida com o uso do *container* Axis 1.4/Java no *host* servidor como *middleware* na comunicação revelou que mesmo com o uso de enlaces com alta capacidade de transmissão e *hosts* de alto desempenho, o *delay* permanecia alto. O gargalo na comunicação entre os *hosts* de domínios distintos ocorria quando as informações atingiam o bloco de comunicação que encaminhava as requisições XML serializadas utilizando SOAP sobre HTTP. O tratamento dos parâmetros fornecidos no bloco de comunicação resultava em atraso considerável em uma simples transferência de *strings*. Esse comportamento foi otimizado com a utilização do *container* Axis2/Java, com a redução do tratamento das informações nos *Web Services*, e com as atribuições de controle de tráfego no domínio interno do WebLab. Esses fatores contribuíram para melhorar a qualidade da oferta de serviços fim-a-fim no domínio do laboratório com tempos de resposta aceitáveis para os usuários dos experimentos. Ao contrário de REST, implementações SOAP fazem referências para recursos no *payload* de suas mensagens, o que dificulta a obtenção de *performance* com *cache servers* ou *proxy servers* de resultados de consultas a URLs. Essa característica reduz a escalabilidade da proposta SOAP porque a aplicação precisa realizar a decodificação da mensagem recebida para ter acesso aos parâmetros e recursos. No entanto, tanto as requisições REST quanto SOAP precisam de aplicações que interpretem a semântica dos documentos recebidos.

Essas considerações reforçam a premissa de que o desempenho da comunicação fim-a-fim entre *hosts* de domínios distintos, como ocorre na Internet, depende não apenas do enlace, mas da qualidade de serviço oferecida por cada um dos nós que encaminham as informações ao longo da rede. De forma semelhante, os testes *DiffServ* mostram a necessidade de uma configuração adequada para reduzir a competição entre os fluxos diferenciados de diferentes experimentos.

Dentre as dificuldades encontradas mereceram destaque:

- o entendimento das configurações *DiffServ*: a elaboração dessas configurações “do zero” é uma tarefa onerosa que foi minimizada com o auxílio do experimento BB. Esse experimento é capaz de manter diversas configurações apresentadas na bibliografia com a criação organizada de disciplinas de fila, classes e filtros. O experimento também simplifica o estudo de diferentes configurações ao gerar os comandos da ferramenta *tc*, priorizando os parâmetros fornecidos. O processo automatizado de remoção também simplifica o estudo de como a alteração de parâmetros nas políticas *DiffServ* afeta o comportamento dos fluxos submetidos ao domínio.
- políticas de segurança da instituição: a Universidade Federal de Uberlândia mantém um *proxy server* que restringe o acesso à rede interna da instituição. Para permitir o acesso aos experimentos do WebLab foi necessário a liberação de portas no *proxy server*.
- restrições de segurança para execução dos experimentos: os experimentos de rede *DiffServ* exigem acesso privilegiado aos *hosts* do WebLab. Os experimentos conferem acesso limitado do usuário a essas configurações, e de forma semelhante, o acesso a esses experimentos é restrito com as permissões mantidas pelos serviços de acesso. O uso de uma *sandbox* Java também impede que alterações não-autorizadas sejam realizadas no *host* do usuário.

- *performance* de acesso ao WebLab: para melhorar o desempenho do acesso e comunicação remota com os experimentos, o servidor do laboratório foi conectado diretamente à rede da instituição com um endereço IP público, sem hosts intermediários atuando como *proxy* na rede do laboratório (como roteadores *wireless*, por exemplo).

6.2 Trabalhos Futuros

O modelo de referência do projeto GigaBOT prevê a integração de WebLabs em uma federação de WebLabs. Essa integração encontra-se em andamento no projeto REALabs na rede KyaTera com o intuito de promover a gerência federada de recursos e a autenticação distribuída de participantes. Mas, para isso, é necessário padronizar tanto a infraestrutura da gerência integrada quanto o processo de desenvolvimento dos WebLabs e de seus experimentos.

A *performance* obtida com a otimização das mensagens XML trafegadas entre o WebLab e o usuário são fundamentais para garantir a qualidade da experimentação remota. O uso de sistemas RESTful aumenta a escalabilidade da representação de recursos e seu mecanismo de controle das mensagens trafegadas otimiza a comunicação. Por isso, essa é uma sugestão de extensão do WebLab.

A criação de experimentos precisa ser documentada para garantir a qualidade do desenvolvimento. A Computação Orientada a Serviços simplifica o processo de criação de experimentos, mas esse processo pode se tornar tão extenso quanto a abrangência da aplicação. Por isso são viáveis soluções de representação dos relacionamentos entre os serviços de interação da arquitetura. Finalmente, os experimentos *DiffServ* poderiam ser estendidos para o uso de aplicações multimídia, tais como VoIP, transmissão de vídeo sobre IP, entre outros.

Trabalhos Publicados pelo Autor

1. Agostinho, L. R., Silveira, D. S., Sousa, R. G., Faina, L. F.: Reconfiguração Dinâmica de Classes DiffServ no Suporte a QoS face à Dinâmica da Rede. In proceedings: *Revista Hífen*, Vol.30, no. 58, p. 129. ISSN 0103-1155. PUCRS Uruguaiana - RS. SIMS 2006.
2. Agostinho, L. R., Sousa, R. G., Faina, L. F., Guimarães, E. G., Coelho, P. R. S. L., Pinto, R. P., Cardozo, E.: Monitoramento e Configuração Dinâmica de Classes DiffServ no Suporte à Qualidade de Serviço. In: *25o. Simpósio Brasileiro de Redes de Computadores - XII Workshop de Gerência e Operação de Redes e Serviços*. Belém do Pará - PA. SBRC WGRS 2007.
3. Agostinho, L. R., Farias, A. F., Faina, L. F., Guimarães, E. G., Coelho, P. R. S. L., Cardozo, E.: Um Framework para Web Labs SOA aplicado em um Domínio de Serviços Diferenciados. In: *XVIII Simpósio Brasileiro de Informática na Educação*. São Paulo - SP. SBIE 2007 (Resumo extendido).
4. Agostinho, L. R., Farias, A. F., Faina, L. F., Guimarães, E. G., Coelho, P. R. S. L., Cardozo, E.: Uma Proposta de Arquitetura para Experimentos DiffServ em Web Labs. In: *XXXIV Conferência Latinoamericana de Informática*. Santa Fé - Argentina. CLEI 2008.
5. Agostinho, L. R., Farias, A. F., Faina, L. F., Guimarães, E. G., Coelho, P. R. S. L., Cardozo, E.: Arquitetura para Experimentos DiffServ em Web Labs utilizando Web Services. In: *III Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica*. Fortaleza - CE. CONNEPI 2008.
6. Agostinho, L. R., Farias, A. F., Faina, L. F., Guimarães, E. G., Coelho, P. R. S. L., Cardozo, E.: NetLab Web Lab: Um Laboratório Remoto de Redes para Experimentos DiffServ. In proceedings: *Revista Hífen*, Vol.32, no. 61. ISSN 1983-6511. PUCRS Uruguaiana - RS. SIMS 2008.
7. Agostinho, L. R., Nunes, R. B., Maia, M.: Suporte a Múltiplas Visões de Software com Módulos Virtuais. In proceedings: *Revista Hífen*, Vol.32, no. 61. ISSN 1983-6511. PUCRS Uruguaiana - RS. SIMS 2008.

Referências Bibliográficas

- [1] S. Fernandes, C. Kamienski, D. Mariz, and D. Sadok. Avaliação de Técnicas de Agrupamento na Amostragem de Tráfego na Internet. *XXIV SBRC*, 2006.
- [2] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A Framework for QoS-based Routing in the Internet. Technical Report RFC 2386, Network Working Group, Agosto 1998.
- [3] B. Teitelbaum, S. Hares, L. Dunn, R. Neilson, V. Narayan, and F. Reichmeyer. Internet2 QBone: Building a Testbed for Differentiated Services. *IEEE Network*, 13(5):8–16, 1999.
- [4] R. Edell and P. Varaiya. Providing Internet access: what we learn from INDEX. *IEEE Network*, 13(5):18–25, 1999.
- [5] K. Nichols, V. Jacobson, and L. Zhang. A Two-bit Differentiated Services Architecture for Internet. Technical Report RFC 2638, 1999.
- [6] P.R.S.L. Coelho, R.F. Sassi, E. Cardozo, E.G. Guimaraes, L.F. Faina, A.Z. Lima, and R.P. Pinto. A Web Lab for Mobile Robotics Education. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1381–1386, 2007.
- [7] D. Lopez-de Ipiña, J. García-Zubia, and P. Orduña. Remote Control of Web 2.0-Enabled Laboratories from Mobile Devices. In *Proc. Second IEEE International Conference on e-Science and Grid Computing e-Science '06*, pages 123–123, Dec. 2006.
- [8] D. Lopez-de Ipiña, J. García-Zubia, and P. Orduña. An Approach for WebLabs Analysis. *Journal of Online Engineering - iJOE*, 3(2), 2007.
- [9] S. Lerman and J. del Alamo. iLab: Remote Online Laboratories, 2005. <http://icampus.mit.edu/projects/iLabs.shtml>. Disponível em 10/05/2008.
- [10] National Instruments. Labview - Laboratory Virtual Instrument Engineering Workbench. <http://www.ni.com/labview>. Disponível em 26/07/2008.
- [11] D. Lopez-de Ipiña, J. García-Zubia, P. Orduña, and U. Hernández-Jayo. Experience with WebLab-Deusto. In *Proc. IEEE International Symposium on Industrial Electronics*, volume 4, pages 3190–3195, 2006.
- [12] E. Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Independent Technology Guides, 2002.
- [13] Y. Yan, Y. Liang, X. Du, H.S. Hassane, and A. Ghorbani. Putting labs online with Web Services. *IT Professional*, 8(2):27–34, 2006.

- [14] A. Agrawal and S. Srivastava. WebLab: A Generic Architecture for Remote Laboratories. In *Proc. International Conference on Advanced Computing and Communications ADCOM 2007*, pages 301–306, 2007.
- [15] N. Simões and M. A. Stanton. A Iniciativa Óptica Nacional e o Projeto Giga, 2002. <http://www.rnp.br/newsgen/0211/giga.html>. Disponível em 06/04/2008.
- [16] E. G. Guimarães, A. T. Maffei, J. L. Pereira, B. G. Russo, M. Bergerman, E. Cardozo, and M. F. Magalhães. REAL: A Virtual Laboratory for Mobile Robot Experiments. In *First IFAC Conference on Telematics Applications in Automation and Robotics*, volume I, pages 209–214, Weingarten, Germany, July 2001.
- [17] E.G. Guimaraes and E. Cardozo. Weblabs sobre redes de alto desempenho. *Revista Fonte*, Ano 4, No. 6, ISSN 1808-0715. 2007.
- [18] E. G. Guimarães, A. T. Maffei, J. L. Pereira, B. G. Russo, E. Cardozo, M. F. Bergerman, and M. F. Magalhães. REAL: A Virtual Laboratory for Mobile Robot Experiments. *IEEE Transactions on Education*, pages 37–42, 2003.
- [19] E. G. Guimarães, E. Cardozo, M. F. Magalhães, M. Bergerman, A. T. Maffei, J. L. Pereira, B. G. Russo, C. A. Miglinski, and R. P. Pinto. Desenvolvimento de Software Orientado a Componentes para Novos Serviços de Telecomunicações. In *XIX SBRC*, 2001.
- [20] E. G. Guimarães. *Um Modelo de Componentes para Aplicações Telemáticas e Ubíquas*. PhD thesis, Faculdade de Engenharia Elétrica e de Computação - UNICAMP, 2004.
- [21] E. G. Guimarães, E. Cardozo, M. F. Magalhães, W. P. Gomes, R. P. Pinto, and L. F. Faina. CCM-tel - Uma Plataforma para Aplicações Telemáticas e Ubíquas. In *XXII SBRC*, 2004.
- [22] W. P. Gomes. Uma Plataforma de Desenvolvimento de Software baseado em Componentes para Dispositivos Móveis. Master's thesis, Faculdade de Engenharia Elétrica e de Computação - UNICAMP, 2005.
- [23] A. F. Farias. Desenvolvimento de um Web Lab SOA no Domínio de Redes de Computadores. Master's thesis, Universidade Federal de Uberlândia, 2008.
- [24] L. R. Agostinho, A. F. Farias, L. F. Faina, E. G. Guimarães, P. R. S. L. Coelho, and E. Cardozo. Uma Proposta de Arquitetura para Experimentos DiffServ em Web Labs. *XXXIV Conferência Latinoamericana de Informática*, 2008. Santa Fé - Argentina. CLEI2008.
- [25] L. R. Agostinho, A. F. Farias, L. F. Faina, E. G. Guimarães, P. R. S. L. Coelho, and E. Cardozo. NetLab WebLab: Um Laboratório Remoto de Redes para Experimentos DiffServ. In *Revista Hifen*, volume 32. PUCRS - Uruguaiana - RS, 2008. ISSN 1983-6511. SIMS2008.
- [26] G. A. Politis, P. Sampatakis, Dr. Iakovos, and S. Venieris. Design of a multi-layer bandwidth broker architecture. In *Lecture Notes in Computer Science; Vol 1938*. Springer Verlag, 2000.
- [27] R. P. Pinto, E. G. Guimarães, E. Cardozo, and M. F. Magalhães. Incorporação de Qualidade de Serviço em Aplicações Telemáticas. *XXI SBRC*, 2003.
- [28] L. Reis and P.R. Guardieiro. An Enhanced Allocation Resource Mechanism for DiffServ Domains. In *Proc. International Conference on Networking and Services*, pages 34–34, 2006.

- [29] C. Bouras, I. Pappas, D. Primpas, and K. Stamos. Using the ns-2 simulation environment to implement and evaluate bandwidth broker models. In *Proc. 2nd Conference on Next Generation Internet Design and Engineering NGI '06*, page 7, Apr. 2006.
- [30] J. Lakkakorpi, O. Strandberg, and J. Salonen. Adaptive connection admission control for differentiated services access networks. *IEEE Journal on Selected Areas in Communications*, 23(10):1963–1972, Oct. 2005.
- [31] C.P.W. Kulatunga, J. Kielthy, P. Malone, and M. O. Foghlu. Implementation of a Simple Bandwidth Broker for DiffServ Networks. In *Proceedings of 2nd International Workshop on Inter-Domain Performance and Simulation*, Mar. 2004.
- [32] Net-SNMP - Simple Network Management Protocol, 2008. <http://www.net-snmp.org>. Disponível em 26/07/2008.
- [33] Projeto TIDIA/KyaTera. Laboratórios de Acesso Remoto sobre Redes Avançadas., 2007. <http://kyatera.incubadora.fapesp.br>. Disponível em 26/07/2008.
- [34] Projeto TIDIA/KyaTera. Fiber-to-the-Lab: Viabilizando a pesquisa colaborativa., 2008. <http://kyatera.incubadora.fapesp.br>. Disponível em 26/07/2008.
- [35] K. J. Ma. Web Services: What's Real and What's Not? *IT Professional*, 7(2):14–21, 2005.
- [36] IBM: International Business Machines. New to SOA and Web Services. <http://www-128.ibm.com/developerworks/webservices/newto/websvc.html>. Disponível em 08/09/2008.
- [37] IBM: International Business Machines. Business Process Execution Language for Web Services version 1.1. www.ibm.com/developerworks/library/specification/ws-bpel. Disponível em 09/08/2008.
- [38] UDDI 101. <http://uddi.xml.org/uddi-101>. Disponível em 06/09/2008.
- [39] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. <http://www.w3.org/TR/ws-arch/>. Disponível em 10/10/2008.
- [40] Sun Microsystems. Remote Method Invocation Home. java.sun.com/javase/technologies/core/basic/rmi/index.jsp. Disponível em 15/05/2007.
- [41] Inc. Object Management Group. CORBA - Common Object Request Broker Architecture. www.corba.org. Disponível em 17/02/2006.
- [42] Microsoft Corporation. COM: Component Object Model Technologies. www.microsoft.com/COM. Disponível em 15/05/2007.
- [43] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), Abril 2007. www.w3.org/TR/soap12-part1.
- [44] W3Schools. SOAP Tutorial. <http://www.w3schools.com/soap>. Disponível em 10/05/2008.
- [45] Apache Foundation. Securing SOAP Messages with Rampart, 2007. http://ws.apache.org/axis2/modules/rampart/1_0/security-module.html. Disponível 05/05/2008.

- [46] W3C Working Group. Web Services Architecture. <http://www.w3.org/TR/ws-arch>. Disponível em 26/08/08.
- [47] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. www.w3.org/TR/wsdl. Disponível em 09/08/2008.
- [48] T. Bray, D. Hollander, A. Layman, and R. Tobin. Namespaces in XML 1.0 (Second Edition). W3C Recommendation 16 August 2006. <http://www.w3.org/TR/REC-xml-names>. Disponível em 08/09/2008.
- [49] C. M. Sperberg-McQueen and H. Thompson. XML Schema. <http://www.w3.org/XML/Schema>. Disponível em 08/09/2008.
- [50] D. C. Fallside and P. Walmsley. XML Schema Part 0: Primer Second Edition. W3C Recommendation 28 October 2004. <http://www.w3.org/TR/xmlschema-0>. Disponível em 08/09/2008.
- [51] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
- [52] K. Chapman. *RESTful Web Services with Apache Axis2*. <http://wso2.org/library/3726>. Disponível em 10/11/2008.
- [53] R. L. Costello. REST (Representational State Transfer). <http://www.xfront.com/sld001.htm>. Disponível em 06/11/2008.
- [54] Apache Foundation. RESTful Web services Support. http://ws.apache.org/axis2/0_94/rest-ws.html. Disponível em 10/11/2008.
- [55] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. Technical Report RFC 1633, 1994.
- [56] J. Wroclawski. The Use of RSVP with IETF Integrated Services. Technical Report RFC 2210, 1997.
- [57] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. Technical Report RFC 3031, 2001.
- [58] K. Nichols, S. Blanke, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC2474. Technical Report RFC 2474, Internet Engineering Task Force, Dec. 1998.
- [59] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. Technical Report RFC 2475, 1998.
- [60] J. F. Kurose and K. W. Ross. *Computer Networking - A Top Down Approach - 4th Edition*. Pearson - Addison Wesley, 2007.
- [61] D. Grossman. New Terminology and Clarification for DiffServ. Technical Report RFC 3260, Internet Engineering Task Force, Apr. 2002.
- [62] A. S. Tanenbaum. *Redes de Computadores*. Editora Campus, 2003.
- [63] L. Balliache. Differentiated Services on Linux HOWTO. <http://www.opalsoft.net/qos/>. Disponível em 26/08/08.

- [64] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. Technical Report RFC 2597, Internet Engineering Task Force, Jun. 1999.
- [65] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. Technical Report RFC 2598, Internet Engineering Task Force, Jun. 1999.
- [66] R. Neilson, J. Wheeler, F. Reichmeyer, and S. Hares. A Discussion of Bandwidth Broker Requirements for Internet2 QBone Deployment, August 1999. cite-seer.ist.psu.edu/neilson99discussion.html. Disponível em 18/08/2008.
- [67] M. A. Brown. Traffic Control HOWTO, 2006. <http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html>. Disponível em 26/08/08.
- [68] Differentiated Services on Linux. <http://diffserv.sourceforge.net>. Disponível em 26/08/08.
- [69] M. A. Brown. Guide to IP Layer Network Administration with Linux, 2007. <http://linux-ip.net/html/linux-ip.html>. Disponível em 26/08/08.
- [70] Linux Advanced Routing & Traffic Control, 2005. <http://lartc.org/>. Disponível em 26/08/08.
- [71] S. Coene. Traffic Shapping with Linux. <http://www.docum.org/docum.org/>. Disponível em 26/08/08.
- [72] L. R. Agostinho, A. F. Farias, L. F. Faina, E. G. Guimarães, P. R. S. L. Coelho, and E. Cardozo. Arquitetura para Experimentos DiffServ em Web Labs utilizando Web Services. *III Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica - CONNEPI 2008*, 2008.
- [73] Inc. 1995-2008 Sun Microsystems. Lesson: Java web start, 2008. <http://java.sun.com/docs/books/tutorial/deployment/webstart/index.html>.
- [74] Inc. 1995-2008 Sun Microsystems. Java web start technology, 2008. <http://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/overview.html>.
- [75] F. Albuquerque. *TCP/IP - Internet: Programação de Sistemas Distribuídos HTML, JavaScript e Java*. Editora Axcel, 2001.
- [76] J. Laine, S. Saaristo, and R. Prior. Rude and Crude, 1999. <http://rude.sourceforge.net>. Disponível em 18/05/2008.
- [77] S. Ubik. Qosplot, 2004. <http://www.ces.net/project/qosip>. Disponível em 18/05/2008.
- [78] Sun Microsystems. Java. www.java.com. Disponível em 14/01/2006.
- [79] D. Benson. *JGraph and JGraph Layout Pro User Manual*. JGraph Ltd., 2004-2006. <http://www.jgraph.com>. Disponível em 10/08/2008.
- [80] K. B. Pham and R. Nguyen. Implementation of a Bandwidth Broker in Java. Master's thesis, School of Electrical Engineering and Telecommunications, 2003.
- [81] B. Adamson and H. Greenwald. Multi-Generator, 2005. <http://cs.itd.nrl.navy.mil/work/mgen>. Disponível em 18/05/2008.

- [82] B. Wilson. GKrellM 2.2.9 - GNU Krell Monitors, 1999-2006. <http://gkrellm.net>. Disponível em 10/05/2008.
- [83] M. Ullah Khan, F. Gutbrodt, and R. Trauter. Model-Driven Development of Real-Time Systems with UML 2.0 and C. *Proceedings of the Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD.MOMPES'06)*, 2006.
- [84] L. R. Agostinho, D. S. Silveira, R. G. Sousa, and L. F. Faina. Reconfiguração Dinâmica de Classes DiffServ no Suporte a QoS face à Dinâmica da Rede. In *Revista Hifen*, volume 30, page 210, PUCRS - Uruguaiana - RS, 2006. ISSN 0103-1155. SIMS2006.
- [85] L. R. Agostinho, R. G. Sousa, L. F. Faina, E. G. Guimarães, P. R. S. L. Coelho, Rossano P. Pinto, and E. Cardozo. Monitoramento e Configuração Dinâmica de Classes DiffServ no suporte à Qualidade de Serviço. *XXV Simpósio Brasileiro de Redes de Computadores - XII Workshop de Gerência e Operação de Redes e Serviços*. WGRS - SBRC 2007.
- [86] M. Devera and D. Cohen. *HTB Linux Queuing Discipline Manual - User Guide*, 2002. <http://luxik.cdi.cz/devik/qos/htb/manual/userg.htm>. Disponível em 10/06/2008.
- [87] Apache Foundation. TCPMon. <http://ws.apache.org/commons/tcpmon>. Disponível em 10/08/08.

Apêndice A

Configuração DiffServ com Policiamento do Tráfego de Ingresso

A configuração *DiffServ* apresentada a seguir ilustra como pode ser realizado o policiamento do tráfego de ingresso nos roteadores de borda com a combinação de diversas disciplinas de fila, classes e filtros oferecidos com a ferramenta *tc* do pacote IPROUTE2. O aplicativo BB organiza essa configuração em tabelas e as exibe em uma estrutura do tipo árvore, simplificando a manutenção e a gerência do controle de tráfego.

```
#!/bin/bash

#1 Policiamento de ingresso no domínio
tc qdisc add dev eth3 handle ffff: ingress

#2 Limitação da banda
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip src 172.16.30.1 police rate 10mbps burst 80k drop flowid 1:0

#3 - AF11
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip tos 0x28 0xff police rate 3mbps burst 80k drop flowid 1:0

#4 - AF21
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip tos 0x48 0xff police rate 2mbps burst 80k drop flowid 1:0

#5 - AF22
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip tos 0x50 0xff police rate 1mbps burst 80k drop flowid 1:0

#6 - BE
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip tos 0x0 0xff police rate 1mbps burst 80k drop flowid 1:0

#7 - EF
tc filter add dev eth3 parent ffff: protocol ip prio 1 u32
match ip tos 0xb8 0xff police rate 3mbps burst 80k drop flowid 1:0

#8 --- qdisc dsmark principal
tc qdisc add dev eth3 handle 1:0 root dsmark indices 64 set_tc_index

#9 --- qdisc htb principal
```

```
tc qdisc add dev eth3 parent 1:0 handle 2:0 htb

#10 --- Classe htb principal
tc class add dev eth3 parent 2:0 classid 2:1 htb rate 10mbps

#[1MB - 3MB]
#11 --- Configuracao especifica da classe AF 1 ---
tc class add dev eth3 parent 2:1 classid 2:10 htb rate 1mbps ceil 3mbps
#12
tc qdisc add dev eth3 parent 2:10 gred setup DPs 5 default 2 prio

#13 --- Classe AF 1 DP (Drop precedence) 1---
tc qdisc change dev eth3 parent 2:10 gred limit 60KB min 20KB max 55KB
burst 40 avpkt 1472 bandwidth 3mbps DP 1 probability 0.02 prio 2
#14 --- Classe AF 1 DP 2---
tc qdisc change dev eth3 parent 2:10 gred limit 50KB min 15KB max 45KB
burst 30 avpkt 1472 bandwidth 3mbps DP 2 probability 0.04 prio 3
#15 --- Classe AF 1 DP 3---
tc qdisc change dev eth3 parent 2:10 gred limit 40KB min 5KB max 35KB
burst 20 avpkt 1472 bandwidth 3mbps DP 3 probability 0.06 prio 4

#[1MB - 2MB]
#16 --- Configuracao especifica da classe AF 2---
tc class add dev eth3 parent 2:1 classid 2:20 htb rate 1mbps ceil 2mbps

#17
tc qdisc add dev eth3 parent 2:20 gred setup DPs 4 default 2 prio

#18 --- Classe AF 2 DP 1---
tc qdisc change dev eth3 parent 2:20 gred limit 60KB min 20KB max 55KB
burst 40 avpkt 1472 bandwidth 2mbps DP 1 probability 0.02 prio 2
#19 --- Classe AF 2 DP 2---
tc qdisc change dev eth3 parent 2:20 gred limit 50KB min 15KB max 45KB
burst 30 avpkt 1472 bandwidth 2mbps DP 2 probability 0.04 prio 3
#20 --- Classe AF 2 DP 3---
tc qdisc change dev eth3 parent 2:20 gred limit 40KB min 5KB max 35KB
burst 20 avpkt 1472 bandwidth 2mbps DP 3 probability 0.06 prio 4

#[1MB - 9MB]
#Para fins de teste, sem respeitar as heurísticas
#21 --- Configuracao especifica da AF 3---
tc class add dev eth3 parent 2:1 classid 2:30 htb rate 1mbps ceil 9mbps
#22
tc qdisc add dev eth3 parent 2:30 gred setup DPs 4 default 2 prio

#23 --- Classe AF 3 DP 1---
tc qdisc change dev eth3 parent 2:30 gred limit 60KB min 20KB max 55KB
burst 40 avpkt 1472 bandwidth 9mbps DP 1 probability 0.02 prio 2
#24 --- Classe AF 3 DP 2---
tc qdisc change dev eth3 parent 2:30 gred limit 50KB min 15KB max 45KB
burst 30 avpkt 1472 bandwidth 9mbps DP 2 probability 0.04 prio 3
#25 --- Classe AF 3 DP 3---
tc qdisc change dev eth3 parent 2:30 gred limit 40KB min 5KB max 35KB
burst 20 avpkt 1472 bandwidth 9mbps DP 3 probability 0.06 prio 4

#[1MB - 2MB]
#Para fins de teste, sem respeitar as heurísticas
#26 --- Configuracao especifica da classe AF 4---
```

```
tc class add dev eth3 parent 2:1 classid 2:40 htb rate 1mbps ceil 2mbps
#27
tc qdisc add dev eth3 parent 2:40 gred setup DPs 4 default 2 grio

#28 --- Classe AF 4 DP 1---
tc qdisc change dev eth3 parent 2:40 gred limit 60KB min 20KB max 55KB
burst 40 avpkt 1472 bandwidth 2mbps DP 1 probability 0.02 prio 2
#29 --- Classe AF 4 DP 2---
tc qdisc change dev eth3 parent 2:40 gred limit 50KB min 15KB max 45KB
burst 30 avpkt 1472 bandwidth 2mbps DP 2 probability 0.04 prio 3
#20 --- Classe AF 4 DP 3---
tc qdisc change dev eth3 parent 2:40 gred limit 40KB min 5KB max 35KB
burst 20 avpkt 1472 bandwidth 2mbps DP 3 probability 0.06 prio 4

#[1MB]
#31 -----Configuracao da fila BE-----
tc class add dev eth3 parent 2:1 classid 2:50 htb rate 1mbps
#32
tc qdisc add dev eth3 parent 2:50 red limit 60KB min 20KB max 55KB
burst 40 avpkt 1472 bandwidth 1mbps probability 0.4

#[3MB]
#33 -----Configuracao da fila EF-----
tc class add dev eth3 parent 2:1 classid 2:60 htb rate 3mbps
#34
tc qdisc add dev eth3 parent 2:60 pfifo limit 5
# Outra possibilidade para EF
#tc qdisc add dev eth3 parent 2:60 tbf rate 2mbps burst 1mb
#latency 100ms peakrate 3mbps minburst 1540

#35 --- Classificador dsmark principal
tc filter add dev eth3 parent 1:0 protocol ip prio 1
tcindex mask 0xfc shift 2 pass_on

# --- Elementos dos classificadores da dsmark principal
#36 --- Filtros da Classe AF 1
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 10 tcindex classid 1:111
#37
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 12 tcindex classid 1:112
#38
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 14 tcindex classid 1:113

#39 --- Filtros da Classe AF 2
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 18 tcindex classid 1:121
#40
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 20 tcindex classid 1:122
#41
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 22 tcindex classid 1:123

#42 --- Filtros da Classe AF 3
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 26 tcindex classid 1:131
```

```
#43
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 28 tcindex classid 1:132
#44
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 30 tcindex classid 1:133

#45 --- Filtros da Classe AF 4
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 34 tcindex classid 1:141
#46
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 36 tcindex classid 1:142
#47
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 38 tcindex classid 1:143

#48 --- Filtro da Classe BE
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 0 tcindex classid 1:1

#49 -- Filtro da Classe EF
tc filter add dev eth3 parent 1:0 protocol ip prio 1
handle 46 tcindex classid 1:50

#50 --- Principal classificador htb
tc filter add dev eth3 parent 2:0 protocol ip prio 1
tcindex mask 0xf0 shift 4 pass_on

# --- Elementos dos classificadores da htb principal
#51 --- Filtro da Classe AF 1
tc filter add dev eth3 parent 2:0 protocol ip prio 1
handle 1 tcindex classid 2:10

#52 --- Filtro da Classe AF 2
tc filter add dev eth3 parent 2:0 protocol ip prio 1
handle 2 tcindex classid 2:20

#53 --- Filtro da Classe AF 3
tc filter add dev eth3 parent 2:0 protocol ip prio 1
handle 3 tcindex classid 2:30

#54 --- Filtro da Classe AF 4
tc filter add dev eth3 parent 2:0 protocol ip prio 1
handle 4 tcindex classid 2:40

#55 --- BE
tc filter add dev eth3 parent 2:0 protocol ip prio 1
handle 0 tcindex classid 2:50

#56 --- EF
tc filter add dev eth3 parent 2:0 protocol ip prio 1
handle 5 tcindex classid 2:60
```

Apêndice B

Implementação dos Experimentos

A implementação dos Serviços de Interação foi padronizada para simplificar a criação de novos experimentos a partir da composição dos serviços disponibilizados. Seguindo o modelo de blocos, a aplicação cliente comunica-se com o *Web Service*. A integridade dessa comunicação é mantida com o uso de uma interface comum entre ambos. Os objetos servidores são instanciados pela fábrica de objetos antes da experimentação e, de forma semelhante, removidos da memória dos hosts do experimento através do processo de *Garbage Collection*. A comunicação entre o *Web Service* e o objeto servidor RMI também é realizada com o uso de uma interface RMI comum a ambos.

B.1 Exemplo de cliente do Web Service BB

```
1 public String addClassHTB(String host, String dev,
2                           String [] campos) {
3     String resultado = "1";
4     try {
5         BBStub stub = new BBStub("http://" + enderecoIPServidor +
6                                   ":8080/axis2/services/BB");
7         BBStub.AddClassHTB servico = new BBStub.AddClassHTB();

8         int i=4;
9         servico.setHost(host);
10        servico.setDev(dev);
11        servico.setParent(campos[i++]);
12        servico.setClassId(campos[i++]);
13        servico.setRate(campos[i++]);
14        servico.setUnidRate(campos[i++]);
15        servico.setCeil(campos[i++]);
16        servico.setUnidCeil(campos[i]);

17        BBStub.AddClassHTBResponse resposta =
18            stub.addClassHTB(servico);
19        resultado = resposta.get_return();
20    } catch (AxisFault e) {
21        resultado = "Falta na comunicacao Axis: " +
22            e.getMessage();
23    } catch (RemoteException e) {
24        resultado = "Excecao remota: " + e.getMessage();
25    }
26    return resultado;
27 } //fim addClassHTB
```

B.2 Exemplo de Web Service

```
1 public String adquirirReferenciaRemota(String host,
2                                     String nomeServico){
3     String resultado = "0";
4     //Adquire uma referência do serviço remoto
5     try{
6         String caminhoServico = "rmi://" + host + "/" + nomeServico;

7         //Obtem a referencia do servico remoto
8         if (nomeServico.equals("TC"))
9             servicoTC = (IServicoTC) Naming.lookup(caminhoServico);
10        else
11            if (nomeServico.equals("BB"))
12                servicoBB = (IServicoBB) Naming.lookup(caminhoServico);
13    } catch ( Exception e ){
14        resultado = "Excecao ao obter a referencia remota: "
15                + e.getMessage();
16    } //fim catch
17    return resultado;
18 } //fim adquirirReferenciaRemota

19 public String addClassHTB(String host, String dev,
20     String parent, String classId, String rate,
21     String unidRate, String ceil, String unidCeil){

22     resultado=adquirirReferenciaRemota(host,"TC");

23     if (resultado.equals("0"))
24     try {
25         //Solicita a execucao remota do metodo
26         //que devera ser realizada pelo objeto servidor remoto
27         resultado = servicoTC.addClassHTB(dev,
28                                     parent, classId, rate,
29                                     unidRate, ceil, unidCeil);
30     } catch ( Exception e ){
31         resultado = "Excecao no cliente RMI: " + e.toString();
32     } //fim catch
33     return resultado;
34 } //fim addClassHTB
```

B.3 Exemplo de Interface de Acesso a Objetos RMI

```
1 public interface IServicoTC extends Remote {
2     ...
3     public String addClassHTB(
4         String dev, String parent, String classID,
5         String rate, String unidRate, String ceil,
6         String unidCeil) throws RemoteException;
7     public String delClassHTB(
8         String dev, String parent, String classID,
9         String rate, String unidRate, String ceil,
10        String unidCeil) throws RemoteException;
11    ...
12 } //fim interface
```

B.4 Fábrica de Objetos RMI - Classe principal

```
1 public static void main(String args[]){
2     //Gerenciador de Seguranca
3     if (System.getSecurityManager() == null) {
4         System.setSecurityManager(new SecurityManager());
5     }
6     try {
7         //Cria o registry na porta especificada
8         Registry registry =
9             LocateRegistry.createRegistry(PORTA_REGISTRO);
10        //Criar uma instancia da fabrica
11        FabricaRMI fabricaRMI = new FabricaRMI();
12        //Registrar o objeto servidor fabricaRMI
13        registry.rebind(NOME_SERVICO_REGISTRADO, fabricaRMI);
14        System.out.println("Fabrica de objetos RMI em " +
15            System.getenv("HOSTNAME") + " ativada.");
16    } catch (Exception e) {
17        System.out.println(e.getMessage());
18    } //fim catch
19 } //fim main
```

B.5 Fábrica de Objetos RMI - método para instanciar objetos

```
1 public String gerenciarServico(
2     String servico, int opcao) throws RemoteException{
3
4     String resultado = "0";
5
6     try {
7         //Localiza o registry na porta especificada
8         Registry registry =
9             LocateRegistry.getRegistry(PORTA_REGISTRO);
10        Remote objetoRegistro=null;
11
12        //Ativar o servico
13        if (opcao == 0){
14            //Criar o servico
15            if (servico.equals("TC")){
16                objetoRegistro = new ServicoTC(); } else
17            if (servico.equals("Rude")){
18                objetoRegistro = new ServicoRude(); } else
19            if (servico.equals("Crude")){
20                objetoRegistro = new ServicoCrude(); } else
21            if (servico.equals("Qosplot")){
22                objetoRegistro = new ServicoQosplot(); } else
23            if (servico.equals("MA")){
24                objetoRegistro = new ServicoMA(); }
25
26        try {
27            //Registrar o servico
28            registry.rebind(servico, objetoRegistro);
29            System.out.println("Servico " + servico + " em "
30                + host + " iniciado.");
31        } catch (Exception e){
32            System.out.println("Excecao ao registrar o servico. "+
```

```
33         e.getMessage());
34         resultado = "1";
35     } //fim catch
36     } else {
37         try {
38             //Remover o servico do registro
39             registry.unbind(servico);
40             //Garbage collection
41             objetoRegistro=null;
42             System.gc();
43             System.out.println("Servico " + servico + " em " +
44                               host + " encerrado.");
45         } catch (Exception e) {
46             System.out.println("Excecao ao remover o servico " +
47                               "do registro. "+e.getMessage());
48             resultado = "1";
49         } //fim catch
50     } //fim else
51 } catch (Exception e ) {
52     System.out.println("Excecao ao localizar o registro. " +
53                       e.getMessage());
54     resultado = "1";
55 } //fim catch
56 return resultado;
57 } //fim gerenciarServico
```

Apêndice C

Monitoramento do Tráfego XML

O desenvolvedor de *Web Services* precisa conhecer a forma como as informações são geradas e encapsuladas em formato XML antes de serem enviadas pela rede. A análise do formato e do conteúdo dessas mensagens permite entender melhor o funcionamento do protocolo SOAP, além de ser material complementar útil para o estudo do desempenho da comunicação. Para o tráfego de pequenas quantidades de informação essa análise pode não ser crucial, mas à medida que mais informações precisam ser enviadas pela rede, tanto o aumento do tráfego de mensagens XML quanto o aumento do tamanho dessas mensagens serão fatores importantes a serem considerados para reduzir o gargalo na rede, o tempo de resposta e indesejáveis *timeouts*.

O monitoramento das mensagens enviadas/recebidas permite orientar o desenvolvedor sobre a correteza das informações contidas no documento XML, além de permitir a análise da estrutura do documento XML gerado. De posse desses dados podem ser realizadas alterações relevantes para otimizar o uso da largura de banda e melhorar a qualidade da comunicação.

Apache TCPMon [87] é uma ferramenta de depuração capaz de monitorar mensagens que trafegam entre duas aplicações que utilizam TCP. A ferramenta também auxilia no monitoramento de mensagens XML ao exibir a descrição e o conteúdo dessas mensagens que trafegam entre a aplicação cliente e o *Web Service*.

Para isso, o TCPMon precisa ser configurado como um *Listener* em um *Target Hostname* e *Target Port*, ou seja, será reservado um endereço IP e uma porta para o TCPMon atuar. Outra opção é a de mantê-lo como um *proxy*. Depois é necessário indicar a *Listen Port* que será utilizada para interceptar as mensagens que trafegam em uma determinada porta.

A Fig. C.1 ilustra o monitoramento de uma mensagem SOAP que envia uma *string* com o conteúdo “Hello World” no *payload* da mensagem. A janela na parte superior da imagem ilustra a descrição da mensagem SOAP e o seu conteúdo que é iniciado com a tag `<?xml version...?>`. O conteúdo é a mensagem SOAP que trafega na rede. A janela na parte inferior da imagem também ilustra a descrição e o conteúdo da mensagem SOAP, mas agora é exibida a mensagem XML retornada pelo *Web Service*. De forma semelhante, o conteúdo da mensagem SOAP de resposta é iniciado com a tag `<?xml version...?>` e a funcionalidade do *Web Service* é a de retornar a mesma *string* enviada pela aplicação cliente, ou seja, “Hello World”.

Na Fig. C.2 foi habilitada a opção REST para o envio e recepção das mensagens SOAP. Observa-se nesse caso a redução do envelope SOAP, mas com a obtenção da mesma *string* de resposta obtida anteriormente. Essa redução ocorreu porque ao utilizar REST as mensagens SOAP são formadas apenas com o trecho XML do corpo da mensagem SOAP original. Como consequência direta obtém-se uma redução significativa na quantidade de informações enviadas à rede e uma redução empírica no tempo de resposta entre sucessivas requisições SOAP. No entanto, a redução do envelope SOAP retira as informações de autenticação, roteamento alternativo, informações sobre quais nós intermediários

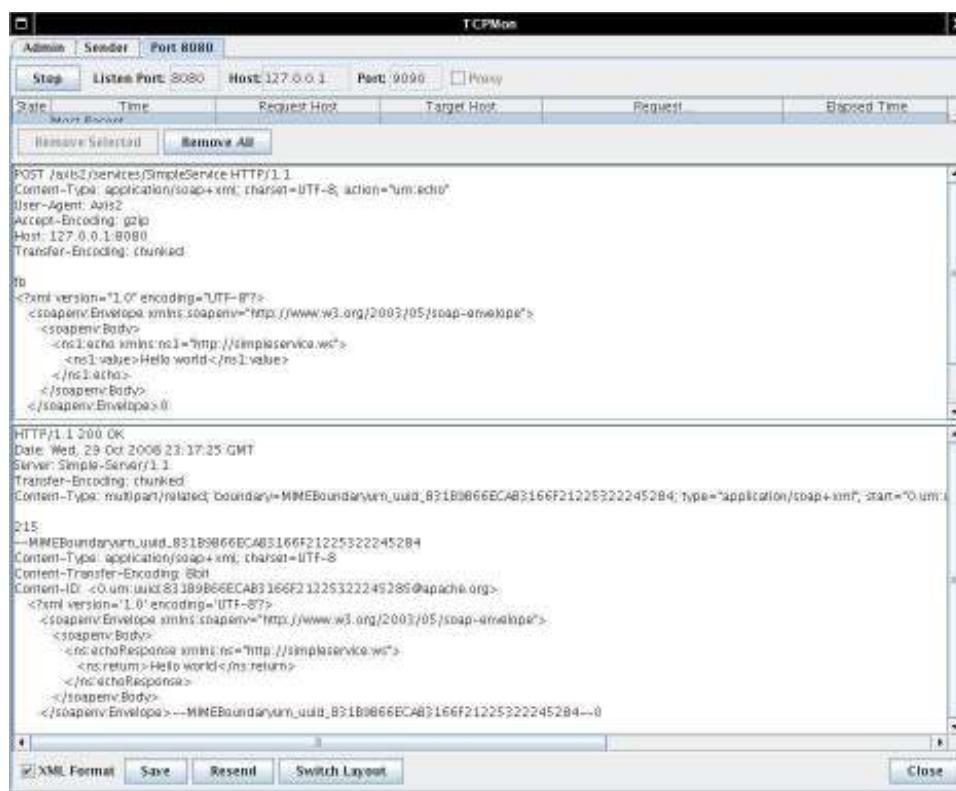


Fig. C.1: Monitoramento de Mensagens SOAP.

podem acessar as mensagens, entre outros, do conteúdo das mensagens.

Para as aplicações em Web Labs que exigem um tempo de resposta reduzido na interação com recursos remotos, a redução do envelope SOAP é uma alternativa viável para otimizar a interação entre o usuário do laboratório e o experimento.

Como alternativa complementar para otimizar o uso da largura de banda entre a aplicação cliente e o *Web Service* é possível compactar as mensagens REST, como ilustrado na Fig. C.3. Nesse caso, uma segurança mínima, mas não suficiente, é conferida às mensagens que trafegam na rede. A mensagem SOAP de resposta é mostrada como na situação anterior porque a compactação não foi habilitada no servidor Web do Axis2. Para a recepção de mensagens compactadas pelo aplicativo cliente do *Web Service* é necessário habilitar a compactação das mensagens de resposta no servidor Web. Em tempo de execução, o *browser* deve ser capaz de descompactar a mensagem e encaminhá-la para o aplicativo cliente correspondente.

A *performance* com a compressão das mensagens XML reduz empiricamente o tempo de resposta entre requisições, mas o servidor também precisa oferecer suporte para a compressão das mensagens de saída (retorno, *output*). O *browser* do usuário, ou o cliente do *Web Service*, deve descomprimir os dados em tempo de execução, reduzindo a quantidade de dados transferidos entre o cliente e o servidor da aplicação. Essa característica é especialmente importante para aplicações assíncronas da Web 2.0, como a tecnologia AJAX, por exemplo.

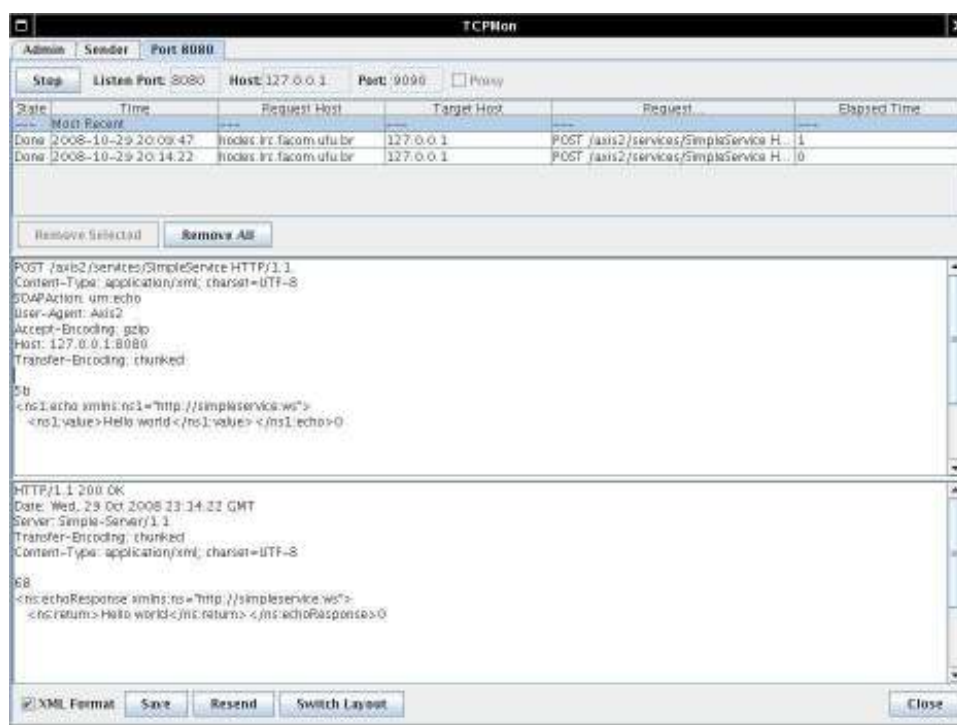


Fig. C.2: Monitoramento de Mensagens XML REST.

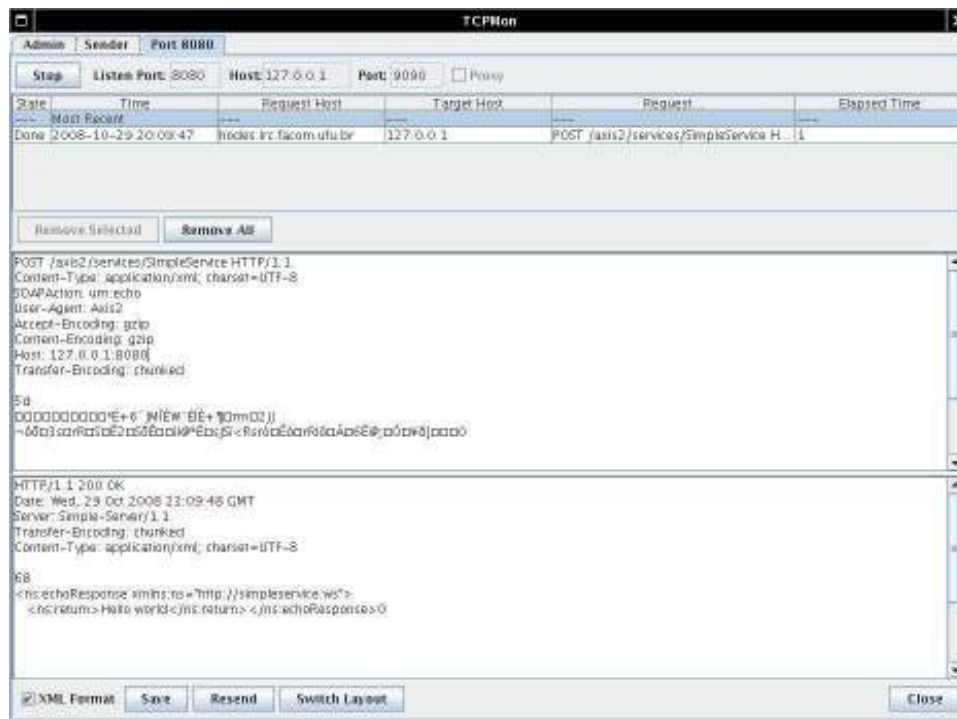


Fig. C.3: Monitoramento de Mensagens XML REST Compactadas.