PROVISIONAMENTO DE RECURSOS EM ARQUITETURA DIFFSERV PARA MELHORIA DA QUALIDADE DE SERVIÇO (QoS)

Marcial Porto Fernandez

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

____

Prof. Aloysio de Castro Pinto Pedroza, Dr.

____

Prof. José Ferreira de Rezende, Dr.

____

Prof. Antônio Carneiro de Mesquita Filho, Dr.d'État

____

Prof. Jorge Lopes de Souza Leão, Dr.Ing.

____

Prof. Julius Cesar Barreto Leite, Ph.D.

____

Prof. Michael Anthony Stanton, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

OUTUBRO DE 2002

FERNANDEZ, MARCIAL PORTO

Provisionamento de Recursos em Arquitetura Diffserv para Melhoria da Qualidade de Serviço (QoS) [Rio de Janeiro] 2002

XXII, 146 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia Elétrica, 2002)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Provisionamento de QoS
2. Gerenciamento de Redes

I. COPPE/UFRJ II. Título (série)

À Irenísia, Raquel e Amanda.

Agradecimentos

Aos meus pais Valentin e Naida (*in memoriam*), pelo amor e apoio em todas as minhas decisões. À minha esposa Irenísia, pelo amor e apoio durante toda a tese. Às minhas filhas, Raquel e Amanda pelo eterno carinho.

Ao Prof. Aloysio de Castro P. Pedroza, por seu incentivo, orientação, reconhecimento e amizade, desde o início desse trabalho. Ao Prof. José Ferreira de Rezende, pela orientação, ajuda e amizade. Aos Profs. Jorge Leão e Michael Stanton, pelas sugestões no Exame de Qualificação. Aos Profs. Antônio Mesquita, Jorge Leão, Julius Leite e Michael Stanton, pela presença na banca e contribuição à tese.

A Luís Henrique, "Baiano", Marcos, Pedro, Vidal, Belém, Arthur, Aline, Kleber, Saulo, Valentim, Gardel, Fagundes, Rubi, Marcio, Eric e Bernardo, pela amizade. E ao pessoal da secretaria, Solange, Bia, Evelyn, Roberto, pelo apoio operacional e amizade.

Ao PEE/COPPE/UFRJ, pelas instalações e equipamentos utilizados.

Ao CNPq, Capes e Anatel pelo auxílio para viagens aos congressos.

Ao CNPq, pelo financiamento da pesquisa de tese.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências(D.Sc.)

PROVISIONAMENTO DE RECURSOS EM ARQUITETURA DIFFSERV PARA MELHORIA DA QUALIDADE DE SERVIÇO (QOS)

Marcial Porto Fernandez

Outubro/2002

Orientadores: Aloysio de Castro Pinto Pedroza
José Ferreira de Rezende

Programa: Engenharia Elétrica

A maioria dos trabalhos na área de Serviços Diferenciados (DiffServ) trata a garantia de Qualidade de Serviço (QoS) em cada nó da rede, supondo-se que a garantia individual proporciona melhor qualidade de serviço em todo o domínio DiffServ(DS). Quando há agregações de fluxos, no entanto, isso pode falhar. Assim, é necessário definir mecanismos de provisionamento dinâmico de recursos, para ajustar a rede de acordo com a variação de tráfego. Um controlador baseado em lógica fuzzy, que reconfigura os nós de um domínio de acordo com o tráfego entrante, pode suprir essa necessidade, devendo os parâmetros do controlador serem definidos em função de contratos administrativos (SLAs). Uma técnica utilizada para coordenar a configuração no domínio é o Gerenciamento Baseado em Políticas, capaz de controlar uma rede heterogênea. Esse trabalho propõe uma metodologia para especificar um controlador fuzzy, partindo de políticas administrativas, e otimizá-lo através do algoritmo Wang-Mendel e algoritmo genético para obter a QoS desejada. As funcionalidades do modelo são demonstradas através de simulação de uma aplicação de Telefonia IP cruzando um domínio DiffServ.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

PROVISIONING RESOURCES IN DIFFSERV FRAMEWORK TO IMPROVE QUALITY OF SERVICE (QOS)

Marcial Porto Fernandez

October/2002

Advisors: Aloysio de Castro Pinto Pedroza

José Ferreira de Rezende

Department: Electrical Engineering

Most of work in the Differentiated Services (DiffServ) area handles Quality of Service (QoS) guarantees on a per node basis, assuming this provides QoS in the whole domain. But it could fail with data flow aggregation. Then, provisioning techniques should be used to reconfigure node parameters according traffic changes. We showed a proposal to use a fuzzy controller that reconfigures all DS nodes according to ingress traffic. Nevertheless, it should be necessary to define the controller parameters according administrative agreement (SLAs). One technique to organize the node configuration could be the Police Based Management, that offer resource to control a heterogeneous network. This work proposes a methodology to specify a fuzzy controller from administrative policies and optimize the fuzzy controller parameters using the Wang-Mendel and a genetic algorithm to achieve the desired QoS. The functionality of a prototype are demonstrated by simulation of a IP Telephony application crossing a DiffServ domain.

Lista de Acrônimos

AIMD : *Additive Increase, Multiplicative Decrease*;

AF : Encaminhamento Assegurado (*Assured Forwarding*);

AG : Algoritmos Genéticos;

CBR : Taxa de Bits Constante (*Constant Bit Rate*);

COPS : *Common Open Policy Service* ;

DS : Serviços Diferenciados (*Differentiated Services*);

EF : Encaminhamento Expresso (*Expedited Forwarding*);

FIFO : *First In, First Out*;

FLC : Controlador em Lógica Fuzzy (*Fuzzy Logic Controller*);

IETF : *Internet Engineering Task Force*;

IIS : Serviços Integrados na Internet (*Internet Integrated Services*);

IP : *Internet Protocol*;

ISO : *International Organization for Standardization*;

MIB : *Management Information Base*;

ns : *Network Simulator*;

OSI : *Open Systems Interconnection*;

OTcl : *Object Tool command language*;

PHB : Comportamento por nó (*Per-Hop Behavior*);

QoS : Qualidade de Serviço (*Quality of Service*);

RTT : *Round-Trip Time*;

SLA : Acordo de Nível de Serviço (*Service Level Agreement*);

SLS : Especificação de Nível de Serviço (*Service Level Specification*);

SGBP: Sistema de Gerenciamento Baseado em Políticas;

SNMP : *Simple Network Management Protocol*;

TCP : *Transmission Control Protocol*;

UDP : *User Datagram Protocol*;

WFQ : *Weighted Fair Queuing*;

WRR : *Weighted Round-Robin*;

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Códigos

Capítulo 1

Introdução

A ARQUITETURA atual da Internet não oferece as garantias de qualidade de serviço (QoS) exigidas pelas aplicações multimídia. A Diferenciação de Serviços (DiffServ) [1] é uma das propostas de solução desse problema, consistindo em prover serviços diferenciados para as agregações de fluxos de dados. Outras alternativas são a Integração de Serviços (IntServ), que assegura QoS para cada conexão, e a Engenharia de Tráfego (Trafic Engineering), onde as tabelas de roteamento nos equipamentos de rede, que incluem métricas adicionais de QoS, determinam o melhor caminho que um determinado pacote deve seguir para a obtenção das garantias solicitadas.

A arquitetura Diffserv, não obstante suas características, pode apresentar problemas quando se trata de oferecer garantias de QoS em todo o domínio ou entre diversos domínios, comprometendo a QoS fim-a-fim. Como não oferece QoS para cada fluxo de dados, a exemplo do IntServ, a qualidade do serviço pode sofrer degradação quando ocorrer congestionamento na agregação de vários fluxos. Além disso, o controle, exercido por nó, pode ser afetado pela heterogeneidade de equipamentos, configurações e topologias, dificultando a garantia de qualidade borda-a-borda em um domínio.

A característica aleatória da chegada de fluxos em diferentes classes de serviço obriga à utilização de alguma técnica para garantir a QoS na arquitetura DiffServ. As principais técnicas são: superprovisionamento e provisionamento dinâmico. O superprovisionamento consiste em dimensionar canais de comunicação acima da demanda real, propiciando QoS pelo fato de que os pacotes encontram uma rede vazia. A grande vantagem do superprovisionamento é a facilidade de implantação, pois aproveita a infraestrutura existente, apenas aumentando a velocidade. A característica dessa técnica é que, normalmente, não há classes de serviços diferentes e todos os fluxos desfrutam do mesmo recurso e QoS. A desvantagem é que manter um canal de comunicações com capacidade acima da demanda produz um aumento de custo, induzindo maiores tarifas na prestação do serviço.

O provisionamento dinâmico consiste em utilizar canais de comunicação compatíveis com a demanda e executar mecanismos de reconfiguração que ofereçam a QoS desejada para determinados fluxos. A grande vantagem é que há um aproveitamento maior da capacidade da rede, e que pode oferecer um serviço de melhor qualidade mantendo a infraestrutura dimensionada de acordo com a demanda. Assim podemos dizer que o provisionamento dinâmico pode oferecer QoS com um custo menor. A desvantagem desse mecanismo é que exige alteração nos equipamentos de rede além de introduzir uma complexidade adicional, dificultando sua gestão.

Em virtude da complexidade dos mecanismos de provisionamento dinâmico, a maioria dos operadores de rede de telecomunicação tem preferido superdimensionar os recursos para obter a QoS desejada. Esse procedimento, no entanto, apresenta um custo muito alto, tanto pela capacidade não utilizada na maioria do tempo (deve-se provisionar pelo pico) como a necessidade de planejar o crescimento, já que construir infra-estrutura de telecomunicações exige uma estimativa de tráfego futuro, que tende a ser imprecisa.

A utilização de mecanismos de provisionamento dinâmico nos nós não são suficientes para garantia a QoS em todo um domínio. É necessário executar o controle de provisionamento no nó Diffserv com uma ação de gerenciamento sobre todo o domínio, isto é, no conjunto de equipamentos do operador da rede de telecomunicação. O gerenciamento baseado em políticas tem sido proposto como infra-estrutura para garantir QoS nas arquiteturas Diffserv e IntServ[2]. Essa técnica não se aplica ao caso de vários domínios, que normalmente estão sob a responsabilidade de entidades administrativas distintas. Entretanto, o gerenciamento é perfeitamente aplicável a um domínio, geralmente controlado por uma instituição apenas.

1.1 Proposta de trabalho

Este trabalho propõe o controle de QoS, dentro de um único domínio Diffserv, apresentando, para isso, um controlador fuzzy, que reconfigura dinamicamente os nós, através do mecanismo de gerenciamento baseado em políticas. A lógica fuzzy foi utilizada em função das características de incerteza e imprecisão inerentes ao fluxo de dados, tendo em vista que a previsão do tráfego em determinado nó da Internet é um problema complexo. A utilização do controlador fuzzy justifica-se pela não linearidade e a ausência de um modelo matemático preciso para tratar estimativas de tráfego[3]. Comparado a um controlador convencional (digital simples), o controlador fuzzy apresenta vantagens significativas no tratamento de variáveis imprecisas, mantendo sua complexidade baixa. A utilização desse tipo de controlador para garantir QoS em uma rede foi apresentada por Vasilakos e Anagnostakis[4]. O trabalho destes autores, no entanto, foi aplicado em ambiente de Engenharia de Tráfego, com concepção bem distinta do ambiente de gerenciamento baseado em políticas.

O controlador fuzzy foi especificado através da definição das variáveis semânticas (função de pertinência) e do conjunto de regras. A alteração nesses parâmetros possibilita alterar o comportamento do controlador, de maneira a tornar sua ação mais ou menos agressiva. Pelo fato dessas variáveis serem definidas de forma abstrata, não podemos garantir a correção e a otimização do controlador fuzzy para o problema proposto[3]. Para melhorar o funcionamento do controlador fuzzy foram utilizadas técnicas baseadas em algoritmos genéticos, que otimizam os parâmetros do controlador fuzzy[5, 6, 7].

Para validar o modelo de controlador proposto foi realizada simulação de um protótipo, através de uma situação prática, com avaliação de tempo de retardo, variação do retardo e descarte de fluxos de Telefonia IP, concorrendo com tráfegos não sensíveis ao retardo.

1.2 Trabalhos relacionados

Recentemente, têm sido apresentadas várias propostas de mecanismos de provisionamento dinâmico para melhorar a QoS. O Projeto Tequila (Traffic Engineering for Quality of Service in Internet, at Large Scale)[8, 9, 10] visa investigar técnicas de provisionamento, controle de admissão e gerenciamento dinâmico de recursos em uma arquitetura DiffServ. O seu ponto de partida são as especificações de serviço, SLS (Service Level Specification), implementadas em um sistema de gerenciamento baseado em políticas. O Tequila não prevê, entretanto, mecanismos de provisionamento dinâmico. Na realidade, o provisionamento é ajustado ao longo do tempo, porém com frequência muito baixa, isto é, mensalmente ou anualmente, produzindo uma situação de provisionamento estático.

O RMD (Resource Management of DiffServ)[11, 12] é uma arquitetura para promover reserva de recursos fim-a-fim em um domínio DiffServ. Essencialmente, ele propõe um protocolo de sinalização que informa a todos os equipamentos do domínio a necessidade de cada classe de serviço.

A proposta de Liao e Campbell[13, 14] contempla um algoritmo de provisionamento dinâmico em redes IP, particularmente em um ambiente DiffServ. Seu funcionamento consiste em ajustar o escalonador de pacotes dos nós a partir de medidas de descarte de pacotes. Essa proposta, no entanto, não contempla uma entidade central responsável pela coordenação dos parâmetros de configuração, como um sistema de gerenciamento baseado em políticas.

1.3 Contribuições

A contribuição do presente trabalho foi a proposta de uma metodologia para construir um mecanismo de provisionamento dinâmico baseado em lógica fuzzy, demonstrando sua eficiência para melhorar a QoS em uma arquitetura DiffServ, com alta variação no tráfego de dados. Uma experiência inicial com os resultados de QoS em uma topologia e perfil de tráfego simples foi apresentada em [15, 16]. Uma experiência mais sofisticada em uma topologia e perfil de tráfego complexos, simulando uma situação real, foi mostrada em [17, 18]. A metodologia completa para construir o mecanismo de provisionamento dinâmico, incluindo as fases de otimização, foram apresentadas em [19, 20]. Finalmente, foi produzido um trabalho mostrando toda a metodologia e experiências com várias topologias[21].

1.4 Estrutura do trabalho

Este trabalho encontra-se organizado da seguinte forma: o capítulo 2 apresenta os fundamentos conceituais, mostrando um resumo sobre DiffServ, gerenciamento baseado em políticas, provisionamento de recursos e controlador fuzzy; o capítulo 3 apresenta toda a metodologia para desenvolvimento do controlador fuzzy para a arquitetura DiffServ; o capítulo 4 mostra o ambiente de simulação, ferramentas utilizadas e o modelo de simulação detalhando o controlador fuzzy implementado; o capítulo 5 apresenta as topologias, modelo de tráfego e os resultados obtidos na simulação; e finalmente, o capítulo 6 apresenta as conclusões desse trabalho e sugere temas para trabalhos futuros.

Capítulo 2

Fundamentos conceituais

APRESENTAMOS, nesse capítulo, uma introdução aos conceitos teóricos utilizados neste trabalho. Apresentamos inicialmente, na seção 2.1, uma introdução sobre Serviços Diferenciados (DiffServ). A seguir, na seção 2.2, mostramos algumas propostas para provisionamento de recursos de rede e, na seção 2.3, uma introdução ao Gerenciamento Baseado em Políticas. Apresentamos então, na seção 2.4, um resumo de várias propostas para prover QoS através de lógica fuzzy. Mostramos, uma introdução sobre lógica fuzzy na seção 2.5 e sobre controlador fuzzy na seção 2.6. Finalmente, apresentamos na seção 2.7 uma introdução à otimização de controlador fuzzy com algoritmo genético.

2.1 DiffServ

A Internet, uma rede por comutação de pacote, não foi concebida para oferecer garantias temporais aos usuários, pois tinha como objetivo oferecer apenas uma comunicação confiável. O crescimento da Internet despertou o interesse de criar novos mecanismos que possibilitassem oferecer recursos para permitir aplicações sensíveis ao tempo, particularmente as aplicações multimídia.

Uma das primeiras propostas com essa finalidade foi a Integração de Serviços (IntServ), que permite ao usuário através do protocolo RSVP (*Resource Reservation Protocol*)[22], requisitar uma reserva de banda passante e espaço em buffer ao longo do caminho, para garantir a QoS desejada. Esta proposta, porém, apresenta problemas de escalabilidade, devido a grande quantidade de fluxos individuais que os roteadores de núcleo precisam tratar em uma rede grande.

A Diferenciação de Serviços (DiffServ) [1, 23, 24] é uma proposta de arquitetura para oferecer recursos de QoS, sem o problema da escalabilidade. Neste caso, os fluxos são agregados em classes de serviço, que têm um padrão de QoS específico. Com uma quantidade de classes limitada, a necessidade de recursos computacionais nos roteadores é reduzida pela menor quantidade de estados a tratar.

A identificação da classe de serviço à qual um pacote pertence é feita através da marcação no campo DS - *Differentiated Service*, antigo campo TOS (*Type of Service*) no cabeçalho IP. O campo DS contêm um valor chamado *codepoint* que é associado a cada classe de serviço. O tratamento que uma determinada classe recebe depende de um conjunto de regras aplicadas a essa agregação, que inclui formas de classificação, escalonamento e tratamento na fila. Esse conjunto de regras é chamado PHB - *Per Hop Behavior*, isto é, comportamento por nó. Um operador de rede que oferece serviço DiffServ tem um contrato de serviço SLA com os usuários e deve cumprir parâmetros de QoS para cada pacote que cruza o domínio, isto é, parâmetros como retardo, variação do retardo (*jitter*) e descarte.

A arquitetura DiffServ, porém, não oferece garantias para os fluxos individualmente, podendo haver violações do SLA quando uma agregação de tráfego exceder a capacidade da rede. Isso exige uma preocupação no provisionamento de recursos, para que as violações não ocorram com freqüência. O IntServ trata individualmente os fluxos e não tem esse problema, porém a escalabilidade é seu ponto crítico.

2.1.1 Arquitetura DiffServ

Para evitar o problema de escalabilidade da arquitetura IntServ, na qual os roteadores de núcleo não conseguem tratar uma grande quantidade de fluxos, a arquitetura DiffServ foi dividida em dois tipos de roteadores, de acordo com a sua posição no domínio: de núcleo ou de borda. Os roteadores de borda ficam na fronteira do domínio e têm a função de fazer a comunicação com roteadores de outros operadores de rede de telecomunicações ou clientes. Os roteadores de núcleo encontram-se todos no núcleo da rede, sem contato com roteadores de outros operadores de rede ou clientes, e onde o tráfego e a quantidade de fluxos é maior devido a agregação de trafegos originários de vários roteadores de borda. A figura 2.1 mostra esquematicamente a arquitetura de um domínio DiffServ.

Na arquitetura DiffServ, os roteadores de borda realizam toda a complexidade de classsificação, marcação, suavização e policiamento. Como esses roteadores tratam uma quantidade menor de fluxos, essas funções, computacionalmente intensas,

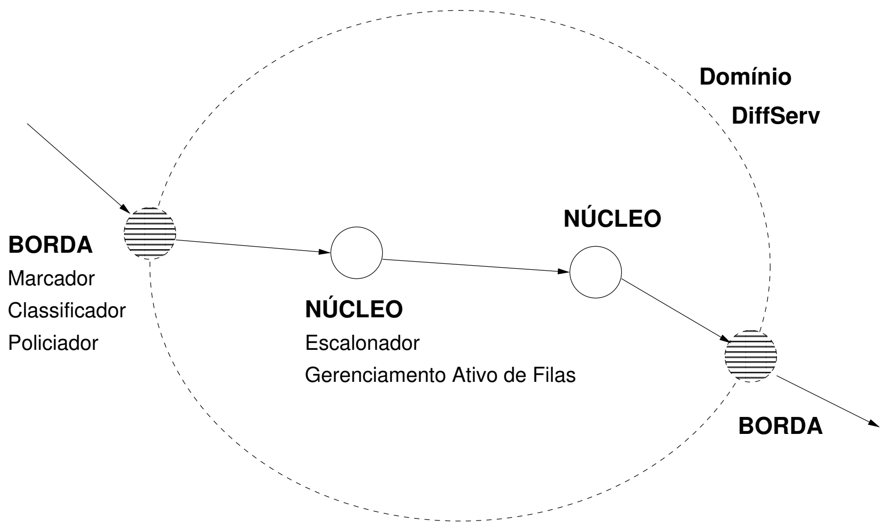


Figura 2.1: Arquitetura DiffServ.

poderão ser realizadas sem prejuízo da escalabilidade.

Os roteadores de núcleo realizam apenas as funções de classificação, escalonamento e gerenciamento de filas. Como o problema de escalabilidade é crítica nos roteadores de núcleo, porque a quantidade de fluxos e as velocidades são grandes, esses roteadores apenas tratam individualmente cada PHB.

2.1.2 PHBs DiffServ

O dois principais PHBs padronizados são: Encaminhamento Expresso *(EF - Expedited Forwarding)*[25] e Encaminhamento Assegurado *(AF - Assured Forwarding)*[26]. Os tráfegos não classificados nessas classes são chamados de Melhor Esforço *(BE - Best Effort)*, isto é, tráfego sem nenhuma garantia de QoS.

O serviço de Encaminhamento Expresso é a classe mais nobre e é adequada para aplicações com requisitos explícitos de retardo e variação de retardo (*jitter*), como telefonia IP e videoconferência. O usuário desse serviço deve especificar no SLA a taxa desejada para seu fluxo de dados e deve ficar ciente que não pode exceder esses limites. Caso o tráfego gerado por esse usuário ultrapasse o SLA contratado, o tráfego excedente deve ser descartado na borda do domínio para não prejudicar os demais usuários do serviço.

O serviço de Encaminhamento Assegurado não oferece garantias explícitas de retardo e variação do retardo, mas oferece garantia de banda passante, mesmo na ocorrência de congestionamentos. É o serviço adequado à aplicações que exigem banda sem restrições temporais como, por exemplo, transferência de arquivos e navegação web. O principal mecanismo dessa classe é o gerenciamento ativo de filas como RED (*Random Early Detection*)[27], que descarta pacotes aleatoriamente antes de ocorrer congestionamento. Esse mecanismo é muito útil para tráfegos FTP e HTTP, pois o controle de congestionamento do TCP reduz a taxa de transmissão quando a conexão perde um pacote, maximizando o aproveitamento do canal (maior taxa útil).

2.2 Provisionamento de recursos

O provisionamento de recursos na rede é um conjunto de operações para oferecer níveis adequados de QoS para cada classe de serviço de forma eficiente. Os mecanismos utilizados para gerenciar o provisionamento incluem controle de admissão, escalonamento, gerenciamento ativo de filas e controle de fluxo.

A QoS oferecida pela rede é uma função dos recursos de rede e da carga aplicada a essa rede. Para oferecer diferentes níveis de QoS para classes diferentes é necessário o controle da carga admitida e da quantidade de recursos necessário para cada classe [28].

A característica aleatória da chegada de fluxos em diferentes classes de serviço obriga à utilização de alguma técnica de reconfiguração desses mecanismos. Mostramos a seguir algumas propostas de mecanismos de provisionamento de recursos, no ambiente DiffServ.

2.2.1 Projeto Tequila

O projeto Tequila (Traffic Engineering for Quality of Service in Internet, at Large Scale) é uma arquitetura para oferecer QoS, proposta por Trimintzios, Pavlou, et al. [8, 9, 10, 29]. O objetivo deste projeto é estudar, especificar, implementar e validar um conjunto de definições de serviço e ferramentas de engenharia de tráfego, de forma a obter garantias de QoS fim-a-fim, mediante dimensionamento cuidadoso, controle de admissão e gerenciamento dinâmico de recursos, em uma rede de Serviços Diferenciados.

A arquitetura Tequila é constituída por dois grupos funcionais. O primeiro grupo inclui o Gerenciador de SLS (Service Level Specification), que é responsável pela assinatura dos usuários e o controle de admissão, além da monitoração dos SLSs. O segundo grupo é responsável pelo gerenciamento de recursos de longo termo (meses ou anos) que realiza o planejamento dos recursos físicos. Analisando a estatística de tráfego ele pode dimensionar os recursos de rede para atender à demanda. Outra função importante desse grupo é o Gerenciador de Políticas, que interpreta as políticas definidas pelo operador da rede, implementa-as nos equipamentos e monitora seu comportamento.

Esse trabalho mostrou a adaptação de um sistema de gerenciamento baseado em políticas a uma rede DiffServ. A coexistência de um sistema baseado em políticas com um sistema tradicional e com um sistema hierárquico foi verificada nessa experiência. Como trabalho futuro, os autores pretendem definir um modelo de informações orientado a objeto, para a arquitetura hierárquica[10]. Além disso, enfocarão o dimensionamento de recursos e o roteamento em função das políticas.

2.2.2 Resource Management in DiffServ (RMD)

A arquitetura RMD, tem por objetivo oferecer reserva de recursos fim-a-fim no domínio DiffServ, conforme proposto por Westberg, Jacobson, et al.[11, 12]. Essa arquitetura inclui novos recursos de reserva à arquitetura DiffServ, como o protocolo *Load Control*, proposto por Westberg, Turanyi, et al.[30]. O protocolo RMD define dois conceitos estruturais: Reserva por Salto (PHR - *Per Hop Reservation*) e a Reserva por Domínio (PDR - *Per Domain Reservation*).

O protocolo PHR é usado no domínio DiffServ no âmbito do nó, como um argumento do PHB (*Per Hop Behavior*), para oferecer reserva de recursos. Este protocolo é implementado em todos os nós do domínio DiffServ. Por outro lado, o protocolo PDR gerencia todos os recursos de reserva no domínio, confiando no estado das reservas de recursos realizados pelo PHR em todos os nós. Esse protocolo somente é implementado nas bordas do domínio (nós de borda). Um protocolo baseado em reserva proposto para a função PHR é o RODA[31].

Os autores citam como vantagens desta proposta a simplicidade e o baixo custo de implementação, além de sua boa escalabilidade.

2.2.3 Algoritmo de Provisionamento em Nós

O Algoritmo de Provisionamento em Nós foi proposto por Liao e Campbell[13, 14] e consiste em uma estratégia de provisionamento dinâmico implementada nos nós de borda e de núcleo. O Algoritmo de Provisionamento de Núcleo previne violações transientes dos SLAs ajustando automaticamente o peso do escalonador e limites de descarte de pacotes nos roteadores de núcleo. O Algoritmo de Provisionamento de Borda tem funcionamento semelhante, apenas atua na agregação do tráfego na entrada da rede. Esses algoritmos não cuidam apenas da justiça entre diversas agregações (classes), mas também dentro de uma mesma agregação onde os pacotes podem tomar caminhos distintos no núcleo da rede.

Esse algoritmo utiliza o conceito de *Utility Function* para balancear a relação justiça-desempenho. Esses trabalhos são uma continuidade dos estudos de negociação interdomínio de Semret e Liao[32, 33] aplicados a um domínio Diffserv. Todas essas propostas se concentram em desenvolver mecanismos de rede, tomando como referência parâmetros absolutos de desempenho. Não está previsto nessa arquitetura uma entidade central para coordenar a configuração dos nós, como um sistema de gerenciamento baseado em políticas.

2.3 Gerenciamento baseado em políticas

O gerenciamento baseado em políticas foi inicialmente proposto por Sloman e Moffett[34, 35]. O autor descreveu uma metodologia para representar parâmetros de comportamento, mais abstratos, a serem cumpridos da melhor maneira possível por cada elemento da rede, levando-se em conta suas características. Em um sistema heterogêneo, constituído por equipamentos de diversos fabricantes, essas políticas poderão assumir valores distintos, em função das características particulares de cada elemento gerenciado. Sloman e Lupu[36] apresentaram uma funcionalidade denominada (*roles*), onde usuários podem substituir outros usuários em determinadas situações. Além disso, um sistema baseado em políticas pode dispor de regras divergentes, devendo o sistema prever um tratamento para conflitos[37].

Uma das aplicações sugeridas para gerenciamento baseado em políticas é o controle da QoS em uma rede IP[38, 39, 40]. Rajan et al[41] dão uma visão geral do uso de políticas em QoS, para serviços diferenciados e integrados. Em Blight e Hamada[2] encontra-se estudo mais detalhado sobre a escalabilidade alcançada com gerenciamento baseado em políticas, para controlar QoS em redes públicas. Flegkas e Trimintzios apresentaram uma arquitetura para gerenciamento de políticas de QoS em redes IP[10].

A arquitetura de um sistema de gerenciamento baseado em políticas, mostrada

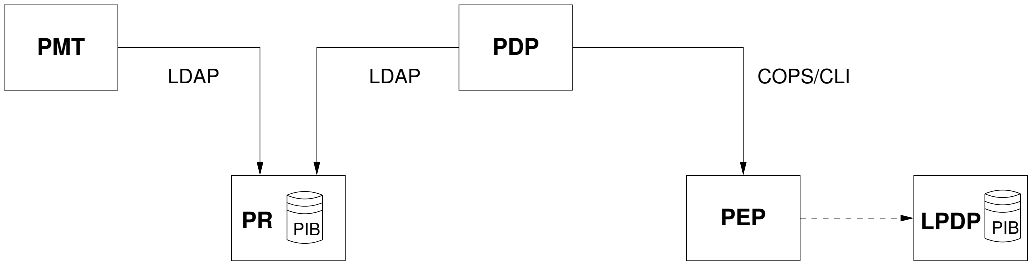


Figura 2.2: Arquitetura de sistema de gerenciamento baseado em políticas.

na figura 2.2, foi proposta pelo IETF [42, 43, 44, 45]. PMT (Policy Management Tool) é a interface com o administrador da rede, que especifica os requisitos de serviço; PR (Policy Repository) é o banco de dados de políticas; PDP (Policy Decision Point) é o componente responsável pela interpretação da política e pela decisão de configuração dos equipamentos, com a função de adaptar a política para cada dispositivo gerenciado; LDAP é o protocolo utilizado para a comunicação entre os diversos componentes; PIB (Policy Information Base) é a tabela que armazena todas as políticas do sistema; COPS [46] é o protocolo utilizado para comunicação com o PEP; PEP (Policy Enforcement Point) é o dispositivo que executa as ações determinadas pelo PDP e LPDP (Local Policy Decision Point) é um elemento opcional que contém seu próprio PDP e PIB, para a operação do sistema na falha do PDP.

Os elementos chave desta arquitetura são o PDP e o PEP, que executam grande parte do processamento de controle do sistema. Nosso trabalho propõe um controlador fuzzy baseado em políticas, que ocupa a função de PDP/PEP na arquitetura de gerenciamento baseado em políticas.

2.3.1 Linguagem de especificação de políticas: Ponder

A linguagem *Ponder*[47] foi proposta por Damianou *et al.*[48] para especificar textualmente políticas de gerenciamento, de acordo com as propostas de Sloman[34] e Lupu[36, 37]. É uma linguagem declarativa orientada a objetos e oferece ao usuário uma interface simples para especificação de políticas, tentando aproximar-se o máximo possível de regras de políticas administrativas.

Essa linguagem define quatro políticas básicas: política de autorização, que pode ser positiva, permitindo o acesso a um determinado recurso, ou negativa, proibindo o acesso; política de obrigação, que exige a execução de determinada ação (previamente autorizada); política de proibição, que proibe a execução de uma ação; e delegação, que permite a um sujeito delegar a outro o controle sobre determinado objeto. Além disso, existem recursos de *composição de políticas*, que possibilitam o agrupamento de políticas para facilitar a descrição de sistemas grandes, ou a *associação de substituição*, onde um usuário pode assumir o papel de outro.

2.4 Lógica fuzzy para prover QoS

É tarefa complexa configurar parâmetros dos equipamentos de rede para manter a QoS, pois requer previsão do tráfego que entrará no sistema em um determinado período. Como o tráfego é uma entidade com características aleatórias, as previsões tendem a ser pouco confiáveis.

Guérin e Orda[49] apresentam um estudo sobre inacurácia e incerteza do estado de uma rede para garantir a QoS das conexões. Este trabalho demonstra que, quando a conexão exige apenas banda passante, a inacurácia não influencia no resultado final, porém, quando a conexão exige garantias de retardo fim-a-fim, a inacurácia da rede torna intratável o processo de seleção do caminho para garantir a QoS. Lorenz e Orda[50] mostram também que a incerteza do estado da rede dificulta a garantia do retardo fim-a-fim em um domínio. No entanto, eles mostram que, decompondo os requisitos de retardo em restrições locais e definindo uma classe de distribuição de probabilidade, é possível estabelecer uma solução eficiente e exata. A complexidade do algoritmo, entretanto, é muito grande para processamento no nó, obrigando então a usar uma aproximação polinomial, que é a proposta desse trabalho.

O problema de incerteza e imprecisão nos remete à solução baseada em lógica fuzzy e realmente vários trabalhos apresentaram aplicações de controladores fuzzy, como é o caso de Li e Nahrstedt[51, 52]. Os autores, no entanto, focaram o ambiente de configuração e não trataram profundamente o controle de recursos na rede. Cheng e Chang [53] mostram um controlador fuzzy para configurar os parâmetros de uma rede ATM. Apesar de tratarem dos parâmetros de rede, pressupõe-se a utilização de rede ATM, que já oferece intrinsecamente recursos de QoS.

Vasilakos e Anagnostakis[4, 54] apresentam um controlador fuzzy para determinar o melhor caminho que um pacote deve seguir, com o objetivo de oferecer garantias de QoS. Esta proposta, aplicável à Engenharia de Tráfego, utiliza algoritmo genético aplicado sobre o histórico do tráfego no nó para definir os parâmetros fuzzy. Pitsillides et al.[55] compararam o desempenho de alocação de banda em redes ATM, usando otimização clássica e via algoritmo genético. Outra comparação de desempenho em redes ATM, com mecanismos fuzzy e convencional, é apresentado por Catania et al.[56, 57]. Ghosh et al.[58] apresentaram um levantamento de aplicações de lógica fuzzy em telecomunicações, mostrando todas as vantagens dessa tecnologia.

2.5 Lógica fuzzy

A lógica fuzzy foi introduzida por Lofti Zadeh[59, 60, 61, 62], como uma generalização da teoria clássica. A extensão sugerida por Zadeh está na possibilidade de um determinado elemento poder pertencer a um conjunto com um valo-r chamado grau de pertinência. Assim, um elemento não simplesmente pertence ou não pertence a um conjunto, como na lógica clássica, mas poderá pertencer a um conjunto com grau de pertinência que varia no intervalo [0,1], onde o valor 0 indica uma completa exclusão, o valor 1 representa completa pertinência e os valores deste intervalo representam graus intermediários de pertinência do objeto com relação ao conjunto. A função que define os graus de pertinência dos elementos é chamada função de pertinência, uma vez que associa para todo elemento do universo de discurso um valor do intervalo [0,1], ao invés do conjunto de apenas dois elementos {0,1}.

A lógica fuzzy utiliza variáveis lingüísticas no lugar de variáveis numéricas. Variáveis lingüísticas admitem como valores apenas expressões lingüísticas, como "muito grande", "pouco frio", "mais ou menos jovem", que são representadas por conjuntos fuzzy. A estratégia de controle de um operador humano pode ser representada como um conjunto de relações condicionais fuzzy que formam um conjunto de regras de decisão. Por exemplo, uma regra típica para controle de temperatura poderia ser: "Se a temperatura está alta e aumentando lentamente, então, aumente o resfriamento um pouco". Esta regra pode ser escrita através de variáveis lingüísticas da seguinte forma: "Se temp = GP e variação_temp = PP então variação_resfr = PP", onde os termos GP e PP significam "grande positivo" e "pequeno positivo".

A teoria da construção de um controlador fuzzy foi mostrada em Lee[3]. A construção de um sistema de controle fuzzy é baseada na idéia de se incorporar "experiência" ou "conhecimento especializado" de um operador humano para se obter a melhor estratégia de controle[63, 64]. Desse modo, a forma das regras empregadas depende do processo a ser controlado.

2.6 Controlador fuzzy

Apresentamos, nesta seção, a arquitetura de um controlador fuzzy apropriado para efetuar controle a partir de variáveis fuzzy. O controlador fuzzy é um mecanismo apropriado para tratar variáveis imprecisas. A figura 2.3 mostra o diagrama de um controlador fuzzy genérico.

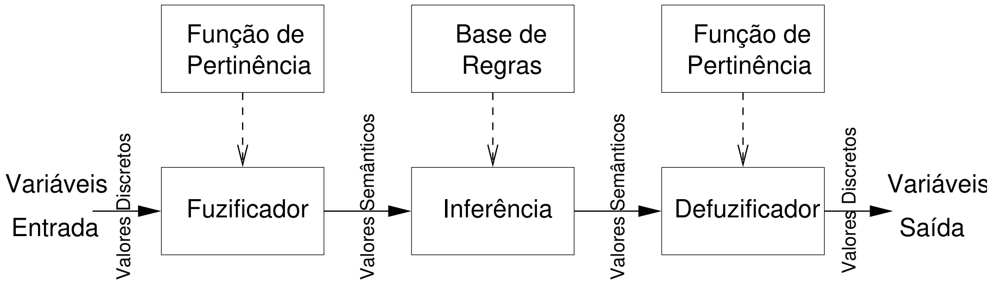


Figura 2.3: Diagrama de um controlador fuzzy.

A entrada de um controlador fuzzy é normalmente um valor discreto, chamado de valor *crisp*. Esse valor discreto é convertido em valor semântico através do mecanismo de fuzificação, utilizando funções de pertinência, detalhadas na seção 2.6.1. A partir desse ponto, o valor discreto é transformado em uma variável semântica, por exemplo "taxa baixa", com um grau de pertinência a essa variável, geralmente um valor real entre 0 e 1, representando respectivamente, total exclusão e total inclusão do valor na variável semântica.

Esse valor é avaliado pelo mecanismo de inferência, que aplica uma base de regras, detalhada na seção 2.6.2. Finalmente o resultado semântico obtido é convertido para um valor discreto (*crisp*) através do mecanismo de defuzificação, mostrado na seção 2.6.3.

2.6.1 Fuzificação e função de pertinência

A lógica fuzzy pode ser vista como uma generalização da lógica booleana convencional. A grande diferença entre as duas está no fato de a lógica fuzzy permitir um certo grau de incerteza, isto é, aceitar a noção de parcialmente verdade ou parcialmente falso. Zadeh[61] enfatiza que a teoria fuzzy não deve ser vista como uma teoria individual, mas como uma metodologia para generalizar qualquer teoria, passando-a da forma discreta para a forma contínua, usando o processo conhecido como fuzificação.

Na lógica fuzzy, os elementos de universo de discurso U são mapeados dentro do intervalo $[0, 1]$. Da mesma forma definem-se os pares ordenados para cada elemento de U e os extremos 0 e 1, que descrevem, respectivamente, a total exclusão e a total inclusão do elemento na variável semântica $\tilde{A}$. Os valores intermediários do intervalo são o grau de pertinência dos elementos em relação à variável $\tilde{A}$. O mapeamento desses valores é, freqüentemente, descrito por uma função de pertinência composta pelo conjunto de variáveis semânticas, representadas pela equação 2.1. A possibilidade de valores intermediários, isto é, um certo grau de incerteza, torna a teoria fuzzy mais apropriada para aplicações reais complexas.

$$\tilde{A}_{T(x)} = \{\,(x,\,\mu_T(x)) \mid x \in U\}\ tq\ \mu_T(x) = \begin{cases} se\ x < a & \mu_T(x) = 0 \\ se\ a < x < b & \mu_T(x) = \alpha.x - \beta \\ se\ b < x < c & \mu_T(x) = \gamma - \alpha.x \\ se\ x > c & \mu_T(x) = 0 \end{cases} \tag{2.1}$$

x Nome da variável

T(x) Valores lingüísticos de x

U Universo de discurso

$\tilde{\mathbf{A}}$ Variável semântica

$\mu_T(x)$ Pertinência da variável x em relação ao valor lingüístico T(x)

A lógica fuzzy usa variáveis lingüísticas ao invés de variáveis numéricas e admite apenas expressões lingüísticas como "alto", "muito alto", "pouco lento", "mais ou menos rápido".

2.6.2 Inferência e base de regras

A grande diferença do controlador fuzzy em relação a um controlador convencional é que o conhecimento pode ser expresso de forma intuitiva, usando regras com variáveis lingüísticas. Um controlador fuzzy apresenta um conjunto de regras semânticas da forma: SE <antecedente1> E <antecedente2> ENTÃO <conseqüente>. Essas regras são mostradas na equação 2.2.

$$\begin{aligned}
&IF\ x_1 = \tilde{A}_1\ AND\ x_2 = \tilde{B}_1\ AND\ x_3 = \tilde{C}_1\ \ \ldots\ \ THEN\ Y = \tilde{Y}_1 \\
&IF\ x_1 = \tilde{A}_2\ AND\ x_2 = \tilde{B}_2\ AND\ x_3 = \tilde{C}_2\ \ \ldots\ \ THEN\ Y = \tilde{Y}_2 \\
&IF\ x_1 = \tilde{A}_3\ AND\ x_2 = \tilde{B}_3\ AND\ x_3 = \tilde{C}_3\ \ \ldots\ \ THEN\ Y = \tilde{Y}_3 \\
&\qquad\qquad\qquad\qquad\qquad\vdots \\
&IF\ x_1 = \tilde{A}_j\ AND\ x_2 = \tilde{B}_j\ AND\ x_3 = \tilde{C}_j\ \ \ldots\ \ THEN\ Y = \tilde{Y}_j
\end{aligned} \tag{2.2}$$

O resultado de cada sentença positiva resultará em um valor de $\tilde{Y}$ e pertinência de $\mu_{\tilde{Y}}$ a serem reunidos através da função definida pela equação 2.3. Esse resultado oferece o valor semântico da resposta que inclui todos os valores obtidos pela base de regras, ponderados pela pertinência de cada variável.

$$\begin{aligned}
\tilde{Y} &= \bigcup_{i=1}^{j} \tilde{Y}_i \\
\mu_{\tilde{Y}} &= min\ \left[\ \mu_{\tilde{Y}_1},\ \mu_{\tilde{Y}_2},\ \mu_{\tilde{Y}_3},\ \ldots\ \mu_{\tilde{Y}_n}\ \right]
\end{aligned} \tag{2.3}$$

2.6.3 Defuzificação e função de pertinência

O valor de resposta semântico não tem valor prático, por isso deve ser convertido em um valor discreto (numérico), que será aplicado ao atuador do controlador. Esse processo é chamado de **defuzificação**. O método mais usual para aplicação em controladores é o do centro de gravidade que consiste em calcular o centro de gravidade da figura obtida através da equação 2.3. O resultado discreto é obtido a partir do valor semântico através da equação 2.4. A necessidade de um cálculo integral, que, poderia ser computacionalmente pesado, em caso de função complexa, não representa problema se o controlador usar funções de pertinência triangulares e trapezoidais.

$$Y^* = \frac{\displaystyle\int_0^1\ \mu_Y(x)\ x\ dx}{\displaystyle\int_0^1\ \mu_Y(x)\ dx} \tag{2.4}$$

2.6.4 Aplicações de controladores fuzzy

Os controladores fuzzy constituem mais importante aplicação da teoria fuzzy. Seu funcionamento é diferente dos controladores convencionais, pois estes exigem o desenvolvimento das equações diferenciais que descrevem o sistema. Em um controlador fuzzy, o conhecimento pode ser expresso de forma intuitiva, usando as variáveis lingüísticas. As aplicações ideais para utilização de controladores fuzzy são as seguintes:

1. Processos muito complexos, onde não há modelo matemático claro;
2. Processos altamente não lineares ou com comportamento probabilístico;
3. Aquelas em que o processamento de conhecimento especializado (formulados lingüisticamente) é o único possível.

2.7 Algoritmo genético

O Algoritmo Genético (AG) são algoritmos de otimização baseados nos princípios de genética e seleção natural. Os AGs foram propostos por John Holland[65] e receberam muitas melhorias desde então. Uma aplicação para a qual AGs têm se mostrado uma ferramenta valiosa é a otimização. Apresentamos, no apêndice D, uma introdução aos conceitos e mecanismos de otimização por AGs.

A otimização de um controlador fuzzy é um processo que busca encontrar a melhor combinação de parâmetros para atingir o melhor resultado possível. Como os parâmetros de um controlador fuzzy como, por exemplo, funções de pertinência, são inter-relacionados, a técnica de AG se mostra bastante útil, pois realiza uma heurística em todo o universo e produz resultados bons em tempo de processamento razoável.

O uso de AGs para otimizar controlador fuzzy tem se mostrado muito útil e aplicável em diversar áreas de utilização. A formalização da metodologia de uso de AGs para otimizar controlador fuzzy foi mostrado por Kim, Moon e Zeigler[6], Velascos e Magdalena[7], Herrera e Lozano[5, 66].

Outra metodologia para otimizar parâmetros de um controlador fuzzy é a de Redes Neurais(RN), mostrado por Arabshahi[67], Altrock[68] e Buja[69]. Entretanto, o resultado obtido através de AG é normalmente mais eficiente[5].

2.8 Comentários

Apresentamos, neste capítulo, uma introdução aos conceitos utilizados em nossa proposta. Fizemos uma introdução à arquitetura DiffServ, mostramos também algumas propostas para provisionamento de recursos de rede e Gerenciamento Baseado em Políticas. Mostramos, ainda, um resumo de várias propostas para prover QoS através de lógica fuzzy. Finalmente, apresentamos introdução à lógica fuzzy, controlador fuzzy e otimização com algoritmo genético.

Com essa visão geral, foi possível contextualizar o presente trabalho entre as várias propostas encontradas na literatura. A seguir, apresentaremos a metodologia proposta, que será detalhadas nos próximos capítulos.

Capítulo 3

Metodologia para provisionamento dinâmico de recursos na Internet

NESSE capítulo, apresentamos a metodologia para construir um controlador para provisionamento de recursos de rede, em um domínio DiffServ. A metodologia tem como ponto de partida a especificação de políticas de gerenciamento para um determinado domínio. A arquitetura proposta, apesar de especificada para DiffServ, pode ser adaptada para qualquer arquitetura baseada em classes de serviço.

Inicialmente, apresentamos a arquitetura de um nó DiffServ, na seção 3.1. Em seguida, na seção 3.2, apresentamos a proposta desse trabalho com a arquitetura do sistema e a metodologia detalhada em seguida. Na seção 3.3, apresentamos a especificação das políticas de QoS; na seção 3.4, mostramos o mapeamento de políticas em parâmetros do controlador fuzzy; na seção 3.5, apresentamos a construção do controlador; na seção 3.6, a seleção dos dados para otimização; e, na seção 3.7, o procedimento de otimização do controlador fuzzy.

3.1 Arquitetura de um nó DiffServ

Mostramos na figura 3.1, o diagrama de um nó de núcleo. Cada nó têm uma fila distinta para cada classe, um classificador que coloca o pacote na respectiva fila e um escalonador que retira pacotes das filas seguindo uma determinada disciplina. Caso a fila esteja cheia, os pacotes que chegarem serão descartados. A fila poderá realizar algum mecanismo de gerenciamento ativo de filas, como por exemplo, o RED[27].

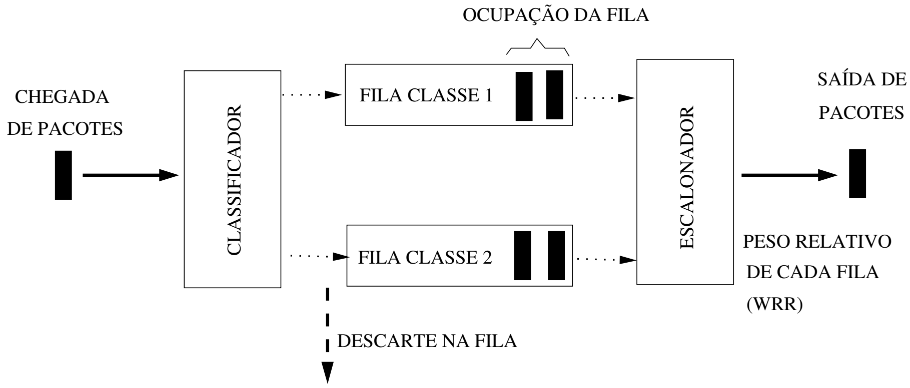


Figura 3.1: Arquitetura de um nó DiffServ de núcleo.

Apresentamos na figura 3.2, o diagrama de um nó de borda. Os nós de borda, além das funções anteriores, contêm um marcador que marca ou remarca cada pacote e um condicionador que tem a função de manter o fluxo de entrada conforme contratado.

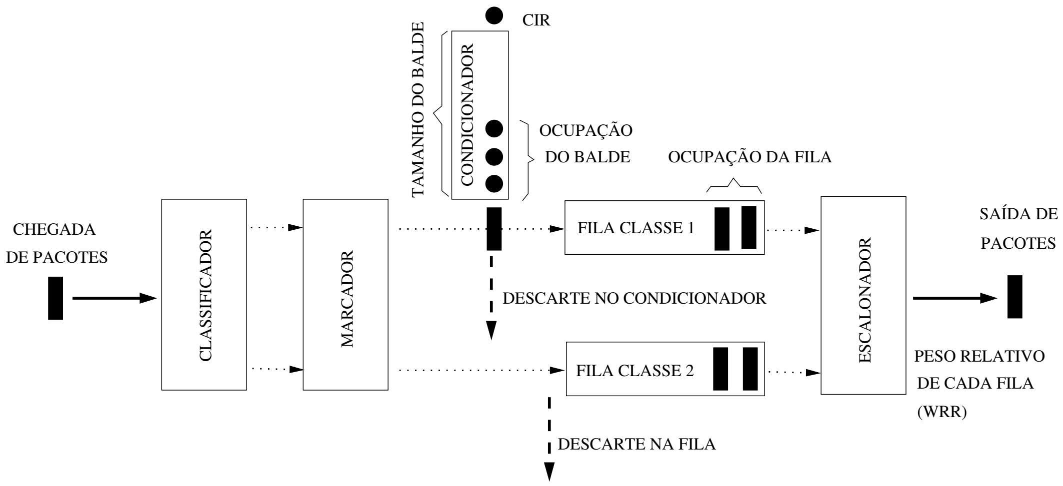


Figura 3.2: Arquitetura de um nó DiffServ de borda.

A arquitetura proposta implementa dois controladores: um para o controle do escalonador, usado nos nós de núcleo e borda, e outro para o condicionador, utilizado apenas em nós de borda.

3.2 Proposta de trabalho

A arquitetura DiffServ visa oferece garantias de QoS na Internet com escalabilidade oferecendo recursos de rede diferenciados para cada classe de serviço. Esta escalabilidade tem seu preço: não podemos garantir a QoS para todos os fluxos de uma mesma classe. Como a Internet tem um comportamento aleatório, devemos estar preparados para detectar violações da QoS. Soluções como IntServ possibilitam o controle por fluxo de dados, garantindo QoS para cada fluxo individualmente, porém, apresentam problemas de escalabilidade nos roteadores de núcleo.

Esse trabalho se propõe a construir um controlador baseado em lógica fuzzy, que implementa um mecanismo de provisionamento dinâmico para melhorar a QoS em um ambiente DiffServ com altos congestionamentos e taxas de perdas. Esse controlador recebe parâmetros de um sistema de gerenciamento baseado em políticas, interpreta essas regras, e reconfigura os nós do domínio para garantir o nível de QoS especificado pelo operador da rede e pelos contratos com os usuários.

3.2.1 Apresentação da arquitetura

Apresentamos na figura 3.3 a arquitetura do sistema proposto. Essa arquitetura tem como objetivo propiciar o melhor desempenho e manter a escalabilidade do sistema.

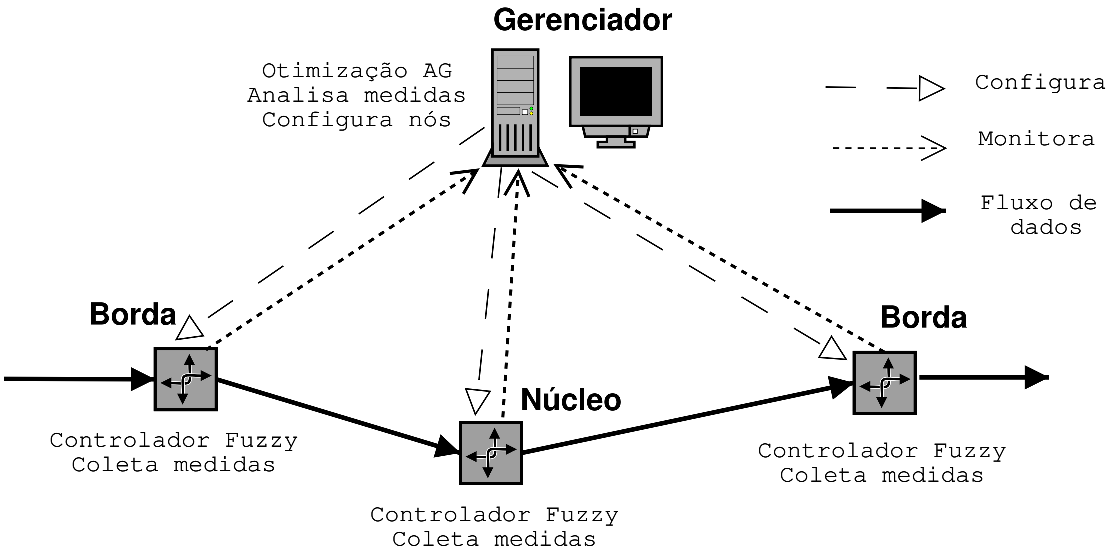


Figura 3.3: Arquitetura do sistema proposto.

Em cada nó do domínio, seja de borda ou núcleo, o controlador realiza medidas do estado atual, calcula o novo valor de configuração, utilizando um mecanismo de lógica fuzzy, e aplica o comando de controle no nó. Como a lógica fuzzy é computacionamente leve, pode ser executada em cada nó do domínio sem interferir muito no desempenho do roteador. O intervalo de operação desse controlador é da ordem de segundos, e portanto, um mecanismo computacionalmente pesado poderia impactar no desempenho global do sistema. Além disso, o nó é responsável por coletar as informações do estado do equipamento (retardo nas filas) e informá-las ao gerenciador, utilizando um protocolo de gerenciamento de redes, por exemplo SNMP (*Simple Network Management Protocol*).

Algumas decisões exigem o conhecimento de um conjunto de nós (domínio), quando então o gerenciador de políticas realiza o cálculo e reconfigura todos os nós necessários. Um exemplo dessa situação é quando o retardo na classe mais prioritária no interior do núcleo aumenta e não existem mais recursos para alocar, obrigando a uma redução na taxa de entrada para não haver descarte no núcleo.

O gerenciador, único no domínio, é responsável por consolidar todas as informações colhidas pelos nós do domínio. Ele também realiza a otimização com algoritmo genético e reconfigura todos os nós regularmente, utilizando o protocolo COPS (*Common Open Policy Service*), por exemplo. Como o algoritmo genético exige uma quantidade maior de recursos computacionais, poderia interferir no desempenho dos roteadores, se fosse executado neles. Lembramos também que a otimização com algoritmo genético ocorre a intervalos maiores, da ordem de horas ou dias, portanto a escalabilidade pode ser mantida mesmo para grandes domínios.

Esse sistema, no entanto, não é apropriado para "corrigir" eventos de falha, como interrupção de linhas de comunicação ou falhas em equipamentos. Essas falhas deverão ser corrigidas pelas metodologias tradicionais de anéis SDH redundantes e roteamento dinâmico. Nosso sistema tem como objetivo manter a QoS fazendo que a rede se reconfigure com alterações na topologia e no perfil de tráfego dos usuários, elementos não tratados nos sistemas tradicionais.

3.2.2 Apresentação da metodologia

Apresentamos, na figura 3.4, o fluxograma completo da metodologia proposta. Nessa figura, os retângulos representam um procedimento da metodologia, que será detalhado nas próximas seções, e os paralelogramos representam as interações com os usuários do sistema ou com os equipamentos de rede. Os operadores de rede de telecomunicações devem cumprir parâmetros de medida de QoS especificadas em contrato, por exemplo, que o retardo máximo da rede seja de 100 ms.

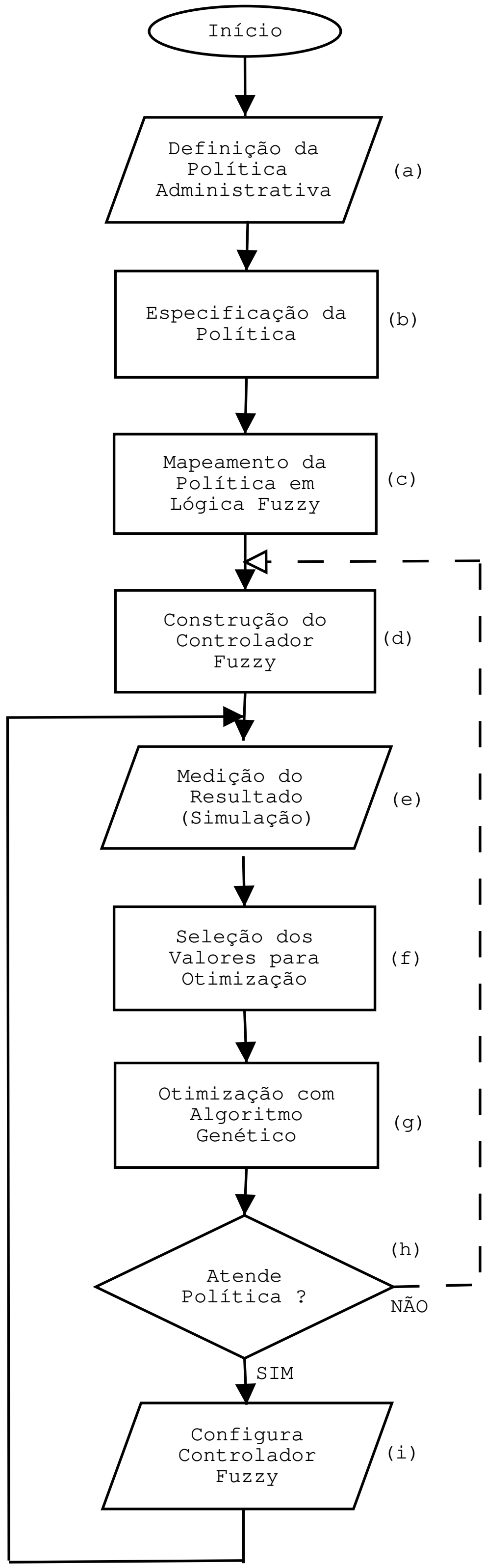


Figura 3.4: Fluxograma da metodologia proposta.

O ponto de partida do presente trabalho são as especificações de políticas administrativas de gerenciamento com base nos requisitos de QoS, conforme figura 3.4(a). A etapa de especificação da política administrativa é mostrada na figura 3.4(b). O detalhamento dessa fase é apresentado na seção 3.3.

Uma política, especificação abstrata de âmbito administrativo, deve ser mapeada em parâmetros de recursos de rede, etapa mostrada na figura 3.4(c). A especificação do controlador tem como base essas políticas administrativas, por isso é apresentada uma proposta de mapeamento de políticas em parâmetros do controlador fuzzy na seção 3.4.

A próxima etapa é a construção do controlador fuzzy, mostrado na figura 3.4(d). Essa etapa define os parâmetros do controlador, como funções de pertinência e base de regras, a partir da especificação de política mapeada na etapa anterior. A construção do controlador é detalhada na seção 3.5.

A definição de regras e parâmetros de um controlador fuzzy não é simples e depende muito da decisão do projetista, podendo, na ocorrência de falhas, produzir resultados errados. Além disso,o atendimento às políticas não garante que esse controlador aproveite o máximo de recurso da rede, isto é, não garante a sua eficiência. A primeira etapa de otimização consiste em construir uma base de regras correta, através do algoritmo de Wang-Mendel mostrado na seção 3.5.3.

Uma vez construído o controlador, podemos aplicá-lo aos equipamentos da rede, ou como em nosso experimento, realizar a simulação. Nessa etapa, mostrada na figura 3.4(e), pode-se coletar os valores de entrada e saída no controlador fuzzy e as medidas de desempenho desejadas como, por exemplo, retardo ou descarte de pacotes.

De posse desses valores, podemos escolher todas as combinações de valores de entrada e saída que maximizam as medidas de desempenho desejadas. Essa etapa é mostrada na figura 3.4(f) e detalhada na seção 3.6. A etapa seguinte consiste na otimização dos parâmetros fuzzy através do algoritmo genético, utilizando como referência de função objetivo os valores selecionados no item anterior. Essa etapa é mostrada na figura 3.4(g) e detalhada na seção 3.7.

A otimização por algoritmo genético pode distorcer as funções de pertinência e regras do controlador, fazendo com que o resultado do controlador deixe de cumprir as políticas originais, apesar de estarem otimizadas para as métricas de escolha dos valores. Na etapa da figura 3.4(h), as novas funções de pertinência e regras são avaliadas com a especificação das políticas. Caso estejam em desacordo, o controlador fuzzy precisa ser refeito na etapa da figura 3.4(d), mostrada com uma linha tracejada. Essa nova reconfiguração do controlador ocorre apenas durante a sua construção inicial e geralmente consiste em limitar as variáveis a serem otimizadas.

Caso o controlador produza um resultado coerente com as políticas originais, ele pode ser usado nos equipamentos para funcionamento normal, conforme mostrado na figura 3.4(i).Sendo assim, definimos uma metodologia para projetar o controlador fuzzy com regras corretas e otimizadas, produzindo um melhor resultado da QoS.

A metodologia admite que o processo de otimização seja realizado continuamente, adaptando os controladores fuzzy de acordo com as mudanças ocorridas na rede (ativação ou desativação de um canal, alteração no padrão de tráfego gerado pelos usuários etc.). Como normalmente essas mudanças são pequenas e eventuais, o algoritmo genético é muito eficiente na otimização. O processo de adaptação é mostrado na figura 3.4, com uma linha cheia, considerando que as políticas administrativas se mantenham inalteradas. Caso contrário, deve-se iniciar a metodologia a partir da etapa da figura 3.4(a).

Comparando com os trabalhos relacionados, apresentados na seção 2.2, nosso trabalho propõe definir um sistema desde a especificação de uma política de gerenciamento até a reconfiguração de parâmetros dos roteadores, ao longo do caminho percorrido pelo fluxo de dados. Nossa proposta é mais completa que o projeto Tequila, apresentado na seção 2.2.1, pois este se concentra na especificação das políticas. Também é mais completa que o RMD, apresentado na seção 2.2.2, pois este trata apenas de um protocolo de sinalização, e o Algoritmo de Provisionamento de Nós, apresentado na seção 2.2.3, que se concentra em implementar mecanismos de reconfiguração de nós, baseado em parâmetros locais absolutos, sem uma entidade centralizadora, como o sistema de gerenciamento baseado em políticas.

3.3 Especificação das políticas de QoS

Podemos definir uma **política** como uma *regra que direciona as opções de comportamento de um sistema de gerenciamento.* A maioria dos trabalhos sobre Gerenciamento Baseado em Políticas focam na especificação de políticas e do modelo de informações das entidades onde as políticas serão aplicadas[70, 45]. Entretanto, pouco trabalho tem sido feito sobre como as entidades interpretam as políticas e definem comandos de configuração nos equipamentos reais. Além disso, ainda falta estudo sobre aplicação de políticas em ambientes grandes, onde interpretações diferentes de uma mesma política podem causar instabilidades e falhas na operação do sistema.

A linguagem **Ponder**[47] foi uma proposta de especificação textual de políticas, que permite a verificação de consistência. Damianou *et al.*[48] a definiu com o objetivo de especificar textualmente políticas de gerenciamento, conforme as propostas de Sloman [34]. A linguagem Ponder é declarativa, orientada a objetos e oferece ao usuário uma interface simples para especificação de políticas, aproximando-se o máximo possível de regras de políticas abstratas.

Essa linguagem oferece cinco comandos principais, indicando as políticas básicas possíveis:

auth+ política de autorização positiva, permitindo o acesso a um determinado recurso;

auth- política de autorização negativa, negando o acesso a um determinado recurso;

oblig política de obrigação, que exige a execução de determinada ação (previamente autorizada);

refrain política de proibição, proibindo a execução de uma ação.

deleg política de delegação do poder de autoridade de um sujeito para outro sobre um objeto gerenciado.

Além das políticas básicas, podemos definir composição de políticas:

group agrupa um conjunto de políticas sob um mesmo nome, podendo ser estabelecida através de apenas um comando;

role permite a substituição do sujeito de um grupo de políticas, onde um usuário pode assumir o papel de outro.

O código 3.1 mostra um trecho da especificação da política do escalonador em *Ponder*. As linhas 4 a 8 definem os valores máximos dos parâmetros de QoS para uma determinada classe de serviço (EF). Podemos ver que o escalonador pode variar de 10% a 90% da banda de saída, o retardo máximo é de 100 ms e o descarte máximo é de 20%. Da linha 11 à 15, são estabelecidas as restrições baseadas nos valores previamente definidos. Nas linhas 17 a 20, são definidos os eventos de disparo das ações, que, em nosso caso, representará uma função de pertinência do controlador fuzzy, conforme metodologia a ser apresentada na seção 3.4.

Código 3.1: Exemplo de especificação Ponder do escalonador

```
   // Especificacao do Controlador do Escalonador
 2 //
   // Define limites minimos e maximos permitidos para o escalonador
 4 const
     minsched = 0.10;  // Escalonador minimo 10%
 6   maxsched = 0.90;  // Escalonador maximo 90%
     MaxDelay = 100;   // Delay maximo 100 ms
 8   MaxDrop = 0.2;   // Descarte maximo 20%

10 // Define condicoes de restricao
   constraint
12   bwMin = bwShare < minsched;
     bwMax = bwShare > maxsched;
14   bwIncrease = bwShare < maxsched;
     bwDecrease = bwShare > minsched;
16
   event   // Definicao dos eventos
18   lowdelay = 0.3 * MaxDelay
     mediumdelay = 0.5 * MaxDelay
20   highdelay = 0.7 * MaxDelay

22   // Autoriza aumentar escalonador se menor que maxsched
     auth+ increaseSchedulerAuth {
24     subject s;
       target t;
26     action increaseBW();
       when bwIncrease;
28   }
     // Obriga aumentar 1 unidade escalonador quando retardo do EF e'
       medio
30   oblig increaseScheduler1 {
       subject s;
32     target t;
       on EF.mediumdelay;
34     do increaseBW(level);
     }
```

A partir desse ponto, as políticas serão executadas pelo sujeito **s** e aplicadas ao alvo **t**. Nas linhas 23 a 28, é especificada uma política autorizando o controlador a executar o aumento da banda (*increaseBW()*), quando a condição particionamento da banda atual (*bwIncrease*) for menor que o máximo permitido (*maxsched*). Nas linhas 30 a 35, é definida uma política determinando que a ação de aumentar a banda (*increaseBW()*), previamente autorizada, seja executada quando o evento retardo na classe EF for médio.

Uma descrição resumida da linguagem *Ponder* é apresentada no apêndice A, com exemplos das especificações comentadas. A especificação da política do controlador do escalonador será apresentada na seção 4.2.1 e da política do condicionador na seção 4.2.2.

3.4 Mapeamento de especificação de políticas em parâmetros de controlador fuzzy

A especificação de políticas traduz uma decisão administrativa em comando do sistema de gerenciamento. Por serem as regras de políticas abstratas e próximas da percepção humana, é muito difícil mapeá-las em regras computacionais, absolutas e exatas por natureza.

A lógica fuzzy tem a característica de tratar variáveis semânticas com certo grau de imprecisão. Por isso, é quase intuitiva a aproximação das regras de especificação de políticas de gerenciamento dos atributos de um controlador fuzzy.

A linguagem *Ponder* permite especificar vários elementos de políticas de gerenciamento de forma textual. As funções restrição e evento podem ser representadas por funções de pertinência, enquanto as funções de comando podem ser representadas pela base de regras. No entanto, ressaltamos que os comandos de autorização e proibição não devem ser mapeados em lógica fuzzy, pois esses comandos são melhor representados por lógica clássica. A única função de autorização que poderia ser mapeada em lógica fuzzy é o comando *role*, onde um usuário pode assumir o papel de outro usuário, o que pode exigir uma avaliação imprecisa, eficientemente tratada pela lógica fuzzy. No entanto, a avaliação desse mapeamento está fora do escopo do presente trabalho.

Apresentamos, na tabela 3.1, um mapeamento de comandos *Ponder* em atributos de lógica fuzzy. A linguagem Ponder, no entanto, não exprime todos os atributos requeridos para a especificação do controlador fuzzy, sendo necessário executar a etapa de otimização, apresentada na seção 3.7, para correto ajuste dos valores do controlador.

Tabela 3.1: Mapeamento de especificação ponder em atributos do controlador fuzzy

Comando Ponder	Atributos do controlador fuzzy
constraint	Centro de gravidade das variáveis lingüísticas mínimo e máximo é igual a esse valor.
event	Os eventos definem a função de pertinência e valor inicial para cada variável lingüística.
oblig	Base de regras na forma Se <condição> então <comando>

3.4.1 Representação do comando constraint e auth

Os comandos **constraint** e **auth-** indicam o limite de restrição de uma autorização, geralmente definindo um valor mínimo ou máximo para certa função. A associação em operação fuzzy consiste em estabelecer o valor defuzificado da variável de saída igual ao valor desejado.

Apresentamos, no código 3.2, uma especificação usando os comandos **constraint** e **auth-**. Nesse caso, a restrição é que a banda mínima do escalonador deve ser 0.10 (ou seja, 10% da banda total). O menor valor atingido por uma função de pertinência é o centro de gravidade (caso este seja o método de defuzificação utilizado) do menor valor semântico da função. O comando **auth-** indica a não autorização de executar o comando de reduzir a banda do escalonador, caso a banda atual seja menor ou igual ao valor mínimo.

Código 3.2: Mapeamento do comando **constraint**

```
constraint
   bwmin = bwShare <= 0.10 ;
inst
   auth- schedulerMin {
      subject s ;
      target sched ;
      action sched.reduceBW() ;
      when bwMin ;
   }
```

A função de pertinência do controlador fuzzy correspondente a essa especificação é mostrada na figura 3.5. Observe que o centro de gravidade do valor semântico "PB" é 0.1. Assim, quando o resultado semântico for apenas "PB" (que é o pior caso) o valor defuzificado será 0.1. Não existe forma de se produzir um valor menor que o da função de menor valor semântico. Poderíamos fazer um mapeamento semelhante para a função peso máximo, que associaria o valor semântico "PA" com o valor defuzificado 0.9.

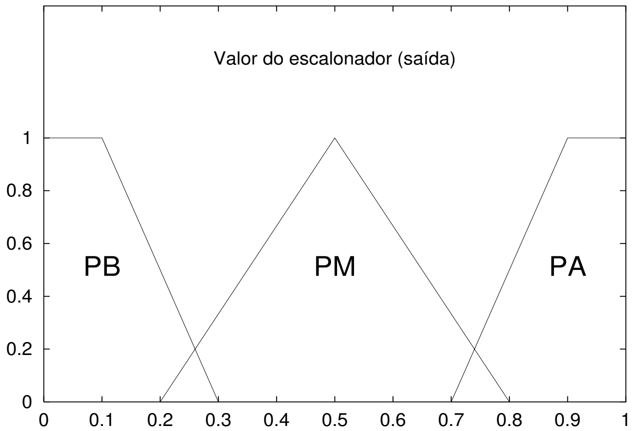


Figura 3.5: Função de pertinência associada a um comando **constraint**

3.4.2 Representação do comando event

O comando **event** lista os eventos que disparam comandos de obrigação (**oblig**) ou proibição (**refrain**). Uma ação pode ter vários eventos possíveis, conforme a política desejada. O mapeamento desse comando é uma função de pertinência de uma variável com seus valores semânticos. Assim, o comando **event** exprime a descrição de uma função de pertinência.

Apresentamos, no código 3.3, uma especificação usando um comando **event** e **oblig**. Atribuímos aos eventos RB, RM e RA, respectivamente, Retardo Baixo, Retardo Médio e Retardo Alto, um valor de disparo. Esses eventos serão utilizados no comando **oblig** como condição de disparo, na linha 10, da ação *sched.increaseBW*, na linha 11.

Código 3.3: Mapeamento do comando **event**

```
1  event
2     RB = 0.1 ;  // Retardo Baixo
3     RM = 0.5 ;  // Retardo Médio
4     RA = 0.9 ;  // Retardo Alto
5
6  inst
7     oblig aumentaEscalonador {
8        subject s ;
9        target sched ;
10       on classeEF.RA ;
```

```
        do sched.increaseBW()
    }
```

A função de pertinência correspondente a essa especificação é mostrada na figura 3.6. Observe que cada valor de evento será associado a um valor semântico da função de pertinência. Podemos notar que a especificação *Ponder* atribui um valor absoluto, sendo o critério de disparo atender ou não a esse valor. Para ser possível o mapeamento, consideramos o valor absoluto como o valor central dos valores semânticos e arbitramos a forma desses valores. O valor exato dessas variáveis não são importantes, pois poderão ser alteradosn no processo de otimização.

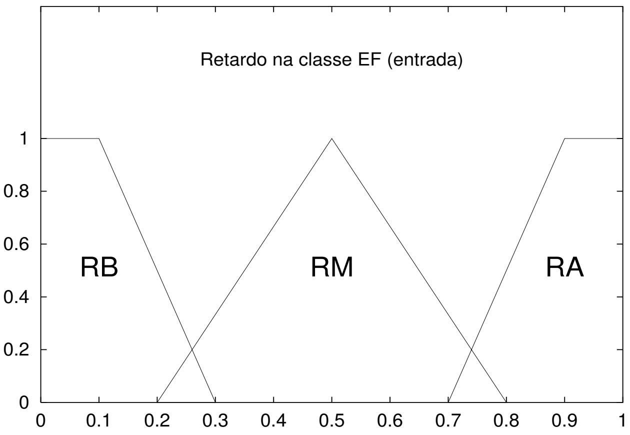


Figura 3.6: Função de pertinência associada a um comando **event**

3.4.3 Representação do comando oblig

O comando **oblig** indica a obrigação de execução de uma ação caso uma determinada condição seja atendida (evento). O mapeamento, portanto, é quase que imediato. O parâmetro **on** do comando **oblig** é mapeado na condição do comando **if** da base de regras, e o parâmetro **do** é mapeado na ação do comando **then**.

Apresentamos, no código 3.4, uma especificação usando o comando **oblig**. Consideramos as mesmas funções de pertinência apresentadas nas figuras 3.5 e 3.6. Nesse comando, caso ocorra o evento retardo médio na classe EF (*classeEF.RM*), é disparada a ação de aumentar a banda do escalonador (*sched.increaseBW()*).

Código 3.4: Mapeamento do comando **oblig**

```
inst
    oblig aumentaEscalonador {
```

```
    subject s ;
    target sched ;
    on classeEF.RM ;
    do sched.increaseBW()
}
```

O código 3.5 apresenta uma descrição JFS da base de regras do controlador fuzzy, a partir da especificação de política apresentada no código 3.4. Podemos observar que o mapeamento de um comando increaseBW() foi desdobrado em várias regras *if* < *condicao* > *then* < *acao* > para abranger todos os valores semânticos da função de pertinência. A partir da ação *increaseBW()* em *Ponder*, definimos três comandos JFS, por ser exigida a especificação de ação para cada valor semântico de entrada. Uma descrição resumida do sistema *JFS* é mostrada no apêndice C

Código 3.5: Código JFS mapeado a partir com comando **oblig**

```
program
   if classeEF RM and sched PB then sched PM ;
   if classeEF RM and sched PM then sched PA ;
   if classeEF RM and sched PA then sched PA ;
```

3.5 Construção do controlador fuzzy

A partir da especificação da política de gerenciamento, podemos especificar o controlador fuzzy. Em nosso exemplo, definimos a política de que toda prioridade deve ser dada à classe EF, isto é, a classe BE será reduzida sempre que houver queda na qualidade da classe EF.

3.5.1 Controlador do escalonador

A regra geral é que se deve dar maior prioridade para a classe EF, quando houver aumento no retardo dos pacotes pertencentes a essa classe. Entretanto a banda passante dedicada à classe BE nunca deve ser menor que 10%, por isso a classe EF nunca deve ocupar mais de 90% da banda. Essa foi a política escolhida para nosso exemplo, porém poderiam ser definidas várias outras, que definiriam novas funções de pertinência e base de regras.

A variável de saída que possibilita o controle do escalonador, depende do tipo do mecanismo utilizado. Quando o escalonador for do tipo prioritário, não há controle a ser efetuado, pois aqui a regra é que as classes menos prioritárias somente serão servidas quando a classe mais prioritária não contiver mais nenhum pacote. O escalonador que permite esse controle deve ser do tipo WRR (Weighted Round-Robin) ou WFQ (Weighted Fair-Queueing), em que as filas são servidas de acordo com o peso definido na configuração. Alterando-se esse peso, podemos modificar a QoS em cada classe.

A primeira variável de entrada é o peso relativo das classes no escalonador, que será usada como referência para a escolha do novo peso. A segunda variável é o tempo que o pacote fica na fila. Como os demais tempos do processamento do pacote são desprezíveis, consideramos o tempo de espera na fila como o tempo total no nó. A terceira variável de entrada é a taxa de descarte da classe BE. Se esse valor estiver alto e o retardo da fila EF for pequeno, isso pode indicar um aumento no peso da fila BE em relação à EF.

Usamos como valor de retardo da classe EF em um nó a média aritmética dos retardos de todos os pacotes que passaram por esse nó, desde a última amostragem do controlador. Um ajuste cuidadoso do período de operação do controlador permite ao valor da média capturar as pequenas variações do retardo.

As variáveis de entrada são:

- Peso relativo no escalonador da fila EF/BE.
- Retardo médio da classe EF.
- Perda de pacotes na classe BE.

A variável de saída é:

- Peso relativo no escalonador da fila EF/BE.

Uma questão importante que merece ser avaliada é como a alteração do valor do valor de configuração (variável de saída do controlador) influencia no funcionamento do roteador. O peso do escalonador define a quantidade de pacotes servidos de cada classe, determinando uma maior ou menor prioridade. Essa alteração não influencia o estado corrente do roteador, sem descartar ou alterando a ordem dos pacotes nas filas. Os efeitos dessa reconfiguração serão sentidos apenas pelos pacotes que chegam ao roteador a partir desse momento.

Exemplo de código do controlador do escalonador

Apresentamos no código 3.6 um exemplo parcial do código do controlador do escalonador na linguagem JFS. Podemos observar a definição do domínio na linha 3, as variáveis de entrada na linha 7 e a variável de saída na linha 10. Nas linhas 13 a 15 definimos os adjetivos da variável *ef_ delay* e, finalmente, na linha 24, a especificação das regras, nesse caso, a partir do algoritmo de Wang-Mendel. Observar que vários valores da função de pertinência são precedidos de %, indicando que esse valor poderá ser otimizado pelo algoritmo genético. Na linha 18, a variável semântica não apresenta indicação de otimização, pois ela define o valor mínimo da saída do escalonador, conforme explicado na seção 3.4.1.

Código 3.6: Exemplo de código JFS do controlador do escalonador

```
   title "Scheduler Controller";
 2
   domains
 4     relative_weight type float "EF/BE" 0.0 1.0;
       delay type float "msec" 0.0 1.0;
 6
   input
 8     ef_delay "EF Delay" domain delay;

10 output
       weight_out "Queue Weight OUT" domain relative_weight defuz ↯
           centroid;
12
   adjectives
14     ef_delay LowDelay 0:1 %0.2:1 %0.4:0;
       ef_delay MediumDelay trapez %0.4 %0.6 base %0.6;
16     ef_delay HighDelay %0.6:0 %0.9:1 1.0:1;

18     weight_out VLP 0:1 0.1:1 0.3:0;
       weight_out LP center %0.3 base %0.3;
20     weight_out MP center %0.5 base %0.3;
       weight_out HP center %0.7 base %0.3;
22     weight_out VHP 0.7:0 0.9:1 1.0:1;

24 program
   extern jfrd input queue_weight_in ef_delay be_discard
26             output weight_out;
```

3.5.2 Controlador do condicionador

O condicionador está presente apenas nos nós de borda do domínio DiffServ. O objetivo principal deste controlador é policiar a entrada de fluxos de dados no domínio, de maneira que os fluxos bem comportados não sejam penalizados. A variável de controle depende do tipo de condicionador utilizado. De forma geral, os condicionadores seguem a filosofia do balde furado, mecanismo constituído por um repositório no qual se colocam fichas a uma taxa constante e do qual cada pacote que entra retira uma ficha. O condicionador policia a taxa de entrada, mas admite rajadas controladas.

A primeira variável de entrada é a taxa de enchimento do balde no condicionador que será usada como referência para a escolha da nova taxa. A segunda variável é o retardo médio dos pacotes na classe EF, no núcleo do domínio. O aumento do retardo no núcleo pode indicar um congestionamento iminente e, provavelmente, descarte na classe EF nos nós de núcleo, o que não é desejado. A terceira variável é o descarte de pacotes da classe EF no condicionador. Quando um pacote chega ao condicionador e não encontra ficha no balde, o mesmo é descartado, pelo que podemos concluir que o tráfego entrante é maior que a taxa permitida.

Em uma situação norma, o condicionador não pode ter a taxa de enchimento do balde alterada durante a operação, pois seu objetivo é manter um tráfego suavizado e conforme ao contratado. Por outro lado, o PHB EF em um domínio DiffServ só deve descartar pacotes na borda[25], sendo indesejáveis os descartes no interior do domínio. Quando não há mais recursos no núcleo do domínio, o descarte é inevitável, e devemos sinalizar para os nós de borda uma redução na taxa de entrada, através da redução da taxa de enchimento do balde. Assim, o valor de retardo médio nos nós de núcleo é enviado ao gerenciador, que sinalizará a redução da taxa de entrada para os nós de borda.

Esta foi a política escolhida para tratar congestionamento no núcleo, entretanto poderíamos usar, dependendo das conveniências do contratante, o critério de manter a taxa contratada e recusar a entrada de novas conexões ou cortar algumas conexões (atuando no marcador).

As variáveis de entrada são:

- Taxa de enchimento do balde.
- Retardo médio na classe EF nos nós de núcleo.

- Perda de pacotes na classe EF no condicionador.

A variável de saída é:

- Taxa de enchimento do balde.

Assim como no controlador do escalonador, a reconfiguração da variável taxa de enchimento do balde, não influencia no estado corrente do roteador. A alteração dessa taxa somente causará mudança após a reconfiguração, não influenciando os pacotes em espera nas filas ou em processamento no condicionador.

3.5.3 Verificação de regras através do algoritmo de Wang-Mendel

A lógica fuzzy apresenta a vantagem de permitir a representação de conceitos ambíguos e produzir uma resposta eficaz mesmo com entradas duvidosas. Porém, um comportamento eficiente requer a definição de regras coerentes. Para que isso não dependa inteiramente do trabalho do projetista do sistema, é desejável a introdução de uma metodologia para produzir funções de pertinência e regras coerentes e eficientes. Utilizamos o algoritmo Wang-Mendel[71], que, a partir do comportamento desejado, cria um conjunto de regras coerentes.

A etapa de verificação consiste na criação de um conjunto de regras corretas, a partir de uma base de conhecimento, constituída por valores numéricos ou semânticos. A implementação utilizada foi apresentada por Cox[72]. Seu funcionamento básico é apresentado no algoritmo 3.1.

Caso a base de conhecimento esteja na forma numérica, os valores serão convertidos para valores semânticos e será construída uma expressão na forma IF ... THEN ....

A maior utilidade dessa metodologia é verificar a consistência do conjunto de regras criadas pelo especialista identificando a ocorrência de regras contraditórias, que levariam a resultados errôneos.

Apresentamos no código 3.7 um exemplo das regras do controlador do escalonador após a verificação realizada com o algoritmo de Wang-Mendel. Observar que em cada linha há uma ponderação dessa regra marcada com % para indicar a possível otimização com algoritmo genético.

Algoritmo 3.1 Algoritmo Wang-Mendel

Entrada: Base de regras não verificada

$contador(i) \leftarrow 0$ {Inicia contador de posto}

$contradicao(i) \leftarrow 0$ {Marca todas as regras não contraditórias}

Normaliza as regras não verificadas na forma IF <variável> IS <adjetivo> AND <variável> IS <adjetivo> AND ... THEN <variável> IS <adjetivo>.

enquanto Existe regra a ser verificada **faça**

  se Regra já existe na base de regras **então**

    Regra não é incluida e $contador(i)$ é incrementado

  senão se Regra ainda não existe e não contradiz nenhuma outra regra **então**

    Regra adicionada à base de regras e $contador(i)$ é incrementado

  senão se Regra contradiz alguma regra existente **então**

    Regra incluída na tabela de regras e marca $contradicao(i) = 1$ e $contador(i)$ é incrementado.

  fim se

fim enquanto

Regras com $contradicao(i) = 0$ são incluídas na base de regras final.

Regras com $contradicao(i) = 1$ inclue a regra com $contador(i)$ maior e descarta a outra.

Código 3.7: Exemplo de código JFS após verificação com Wang-Mendel

```
program
  switch;
  case ef_delay LowDelay;
    switch;
    case be_discard LowDiscard;
      ifw %0.7 queue_weight_in LP then weight_out LP;
      ifw %0.7 queue_weight_in MP then weight_out MP;
      ifw %0.7 queue_weight_in HP then weight_out HP;
    end;
    switch;
    case be_discard MediumDiscard;
      ifw %0.7 queue_weight_in MP then weight_out LP;
      ifw %0.7 queue_weight_in HP then weight_out MP;
      ifw %0.7 queue_weight_in VHP then weight_out HP;
    end;
```

3.6 Seleção dos valores para otimização do controlador fuzzy

O processo de otimização por AG necessita de uma função objetivo, para onde a otimização deve convergir. A grande dificuldade, no entanto, é definir essa função objetivo[5]. No caso do controlador do escalonador, temos como entrada as variáveis peso inicial do escalonador, retardo na fila EF e descarte na fila BE e, como saída, o peso do escalonador. No entanto, a variável medida a ser otimizada é o retardo da fila EF, que somente pode ser avaliada em simulação.

Em vista disso, definimos uma metodologia para obter os valores utilizados na otimização. Durante a simulação com as funções de pertinência arbitradas pelo projetista, armazenamos todas as combinações de parâmetros de entrada do controlador fuzzy e as métricas de avaliação como, por exemplo, o valor de retardo obtido no período seguinte e a taxa de descarte de cada classe, ou seja, o resultado obtido pela aplicação de parâmetros na simulação.

De posse desses valores, escolhemos todas as combinações de parâmetros que produzem um retardo baixo, aquelas que produzem uma maior redução no retardo dos pacotes em medidas consecutivas e aquelas que produzem uma menor taxa de descarte agregado(EF + BE). Foi necessário estabelecer um critério para a ordenação das medidas, pois elas são independentes e algumas até contraditórias, como baixo retardo e baixo descarte. Ordenamos as medidas individualmente por ordem crescente, para a primeira e terceira métrica, e decrescente, para a segunda métrica. Calculamos um valor de posto igual a média da posição de cada parâmetro, isto é, menor retardo absoluto, maior redução no retardo e menor descarte. Como as duas primeiras medidas estão relacionadas ao retardo, podemos dizer que a otimização utilizou uma ponderação de 66% para retardo e 33% para descarte. Assim, escolhemos as melhores combinações de parâmetros para nosso problema, segundo a ponderação escolhida.

Os valores escolhidos são utilizados como base de conhecimento para a aplicação do algoritmo genético, que produzirá o melhor conjunto de parâmetros do controlador fuzzy para atingir a otimização desejada. Utilizamos como referência para escolha os 20% melhores valores. Escolhemos também 10% e 30% dos melhores valores, porém os resultados obtidos foram semelhantes, mostrando que a quantidade de valores escolhida não é importante para a otimização, pelo menos para os percentuais testados.

3.7 Otimização de controlador fuzzy através de algoritmo genético

O controlador definido com funções de pertinência arbitradas pelo projetista e com a base de regras verificado pelo algoritmo de Wang-Mendel, mostrado na seção 3.5.3, garante a definição de um controlador correto, porém ainda sem garantia de eficiência. Para otimizar o resultado do controlador, precisamos escolher os melhores parâmetros do controlador através da utilização do algoritmo genético.

O mecanismo utilizado neste trabalho foi detalhado por Michaelewic[73]. No apêndice D, mostramos uma introdução aos conceitos e mecanismos de otimização por AGs. Apresentamos, na tabela 3.2, um sumário dos mecanismos de funcionamento escolhidos no presente trabalho.

Tabela 3.2: Mecanismos de algoritmos genéticos escolhidos

Tipo	Mecanismo escolhido
Codificação do cromossomo	Real com 4 casas decimais
Funcionamento	Método Simples
Seleção	Método da Roleta
Cruzamento	Multiponto

3.7.1 Funcionamento dos algoritmos genéticos

O algoritmo genético cria inicialmente um conjunto de soluções, a partir dos valores arbitrados inicialmente, para um determinado problema (chamadas de indivíduo). Esse conjunto é chamado de população. O posto de cada indivíduo é calculado pelo teste da solução em relação ao valor desejado. Os indivíduos com posto baixo são substituídos por novos indivíduos criados a partir de indivíduos com posto alto. Esse processo continua até uma solução ser encontrada ou alguma condição de término seja alcançada (por exemplo, número de gerações). O indivíduo com maior posto é a solução encontrada para o problema.

A partir da tabela com valores de entrada e saída desejados, escolhidos na seção 3.6, o algoritmo genético cria um indivíduo, verifica se o resultado calculado se aproxima do valor desejado e pontua cada avaliação. A cada geração, os indivíduos mais aptos são selecionados e combinados (através de *crossover*), obtendo-se um conjunto de parâmetros otimizado para a tabela de valores desejados. Eventualmente, os parâmetros podem convergir para um bom valor local, mas que não é o melhor para o universo. Para evitar isso, aplica-se o processo de *mutação*, incluindo no conjunto de avaliação um valor aleatório totalmente novo, que pode indicar uma nova região de otimização.

Após uma série de iterações, obtemos um conjunto de parâmetros otimizado para o controlador. A grande vantagem desse procedimento é que ele otimiza automaticamente os parâmetros das funções de pertinência, sem depender do critério do projetista.

A vantagem da utilização de algoritmo genético para otimização é que ele pode ser executado continuamente, a partir de medidas de desempenho da rede durante o funcionamento normal. Esse procedimento permite o ajuste dos parâmetros quando ocorrem modificações na topologia ou no padrão de tráfego, durante a operação da rede. A execução do algoritmo genético, no entanto, não provoca queda no desempenho da rede, pois a otimização pode ser realizada em equipamentos dedicados (os roteadores da rede somente executam o controlador fuzzy). Além disso, a otimização não precisa ocorrer em períodos muito curtos, pois as mudanças que poderiam causar alterações ocorrem na ordem de dias ou semanas.

3.7.2 Parâmetros de configuração do algoritmo genético

A configuração correta dos parâmetros do algoritimo genético é, sem dúvida, um dos aspectos mais importantes na estratégia dos AGs. Não existe uma metodologia genérica, uma vez que tais configurações dependem da aplicação a ser otimizada. É importante também levar em consideração os tempos de execução do problema e os recursos computacionais disponíveis. Apresentamos a seguir os parâmetros mais importantes e sua influência no desempenho do algoritmo:

Os parâmetros do algoritmo genético utilizado na otimização do controlador fuzzy estão relacionados na tabela 3.3. Foram usados valores dentro das faixas usuais, conforme comentados em seguida, porém com os ajustes para melhoria da eficiência.

Tabela 3.3: Parâmetros do algoritmo genético utilizados na otimização

Parâmetro	Valor
Tamanho da população	500
Taxa de cruzamento	65%
Taxa de mutação	5%
Critério de parada	90% da população igual
Número de gerações	600

Tamanho da população

O tamanho da população N afeta o desempenho global e a eficiência dos algoritmos genéticos, já que uma população pequena fornece uma cobertura pequena do espaço de busca do problema. Uma grande população fornece uma cobertura maior do espaço de busca, além de prevenir convergências prematuras para soluções locais. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais e tempos mais longos para resolução de problemas simples.

Assim, a escolha correta do tamanho de população, que produz respostas corretas e rápidas, depende do problema e da sua formulação. Geralmente, usa-se um tamanho de população proporcional ao tamanho do cromossomo, isto é, quanto maior for o cromossomo maior deverá ser o tamanho da população, para manter uma diversidade razoável. A literatura sugere populações entre 50 e 100 cromossomos como bom compromisso cobertura-desempenho[74].

O tamanho da população utilizada foi de 500 indivíduos, que possibilitou uma melhor cobertura do espaço, mas que se mostrou eficiente em nossa experiência, pela capacidade computacional disponível.

Taxa de cruzamento

A taxa de cruzamento P_C define a probabilidade de ocorrer um cruzamento. Quanto maior for essa taxa, mais rapidamente novas estruturas serão introduzidas na população. Porém, estruturas com boas aptidões poderão ser substituídas mais rapidamente, perdendo-se boas oportunidades para achar uma solução ótima. Com um valor baixo, o algoritmo pode tornar-se muito lento, demorando a encontrar a solução ideal. A maior parte da literatura recomenda uma taxa de cruzamento entre 50% e 95%[74].

A taxa de cruzamento utilizada foi de 65%, valor usual que se mostrou adequado para nossa experiência.

Taxa de mutação

A taxa de mutação P_M define a taxa de ocorrência de mutação. Uma taxa de mutação baixa diminui a inclusão de indivíduos novos na população, provocando o encontro de resultados baseados em máximos locais, pois restringe o espaço de busca. Uma taxa muito alta transforma a busca do algoritmo genético em busca essencialmente aleatória.

Alguns pesquisadores recomendam a escolha da taxa de mutação com base no tamanho do cromossomo e da população. De Jong [75] sugere uma taxa de mutação inversamente proporcional ao tamanho da população. A maior parte da literatura recomenda uma taxa de mutação entre 0,1% e 5% [74].

Utilizamos um valor de taxa de mutação alta (5%), porém dentro da faixa recomendada. Esse valor se justifica pelo fato de usarmos o método de seleção da roleta, que pode produzir uma baixa diversidade na população e não atingir o valor máximo global da função, conforme comentado na seção D.5.

Número de gerações

O número de gerações define a quantidade de populações criadas até a resposta final. Com um valor baixo, podemos encontrar rapidamente uma solução ótima local, porém ainda longe da solução global. Com um valor alto, temos garantia de encontrar a solução ótima, porém o tempo para chegarmos à resposta pode ser longo, além de, no caso de problemas simples, criarmos novas gerações desnecessariamente, mesmo após a solução ótima ter sido encontrada. O valor ideal para este parâmetro depende do problema e da sua formulação. Em nossa experiência, estabelecemos um limite relativo válido para qualquer problema: encerramos o processamento do AG quando 90% da população for constituída de apenas um indivíduo, que é a solução desejada.

O programa JFS[76] oferece apenas duas formas de encerrar o processamento: pelo tempo decorrido ou pela quantidade de gerações. No entanto, para podermos garantir que encontraríamos o valor de ótimo global, estabelecemos que a população contivesse pelo menos 90% de um mesmo indivíduo. Assim fomos obrigados a executar uma grande quantidade de gerações e testar se o critério de fim tinha sido atingido. Em todas as otimizações realizadas, a quantidade de 600 gerações foi suficiente para atingir essa meta.

3.7.3 Exemplo de código após a otimização com AG

Apresentamos no código 3.8 um exemplo do código do controlador do escalonador após a otimização com o algoritmo genético. Observar que vários valores precedido com % foram alterados em relação ao código 3.7.

Código 3.8: Exemplo de código JFS após otimização com AG

```
adjectives
    ef_delay LowDelay 0:1 %0.2:1 %0.22:0.0004;
    ef_delay MediumDelay base %0.6 trapez %0.3 %0.5;
    ef_delay HighDelay %0.6:0 %0.7513:1 1:1;

    weight_out VLP 0:1 0.1:1 0.3:0;
    weight_out LP %0.35 base %0.3;
    weight_out MP %0.57 base %0.31;
    weight_out HP %0.76 base %0.3;
    weight_out VHP 0.7:0 0.9:1 1:1;

program
    switch;
    case ef_delay LowDelay;
      switch;
      case be_discard LowDiscard;
        ifw %0.7 queue_weight_in LP then weight_out LP;
        ifw %0.4793 queue_weight_in MP then weight_out MP;
        ifw %0.4152 queue_weight_in HP then weight_out HP;
      end;
      switch;
      case be_discard MediumDiscard;
        ifw %0.3236 queue_weight_in MP then weight_out LP;
        ifw %0.2917 queue_weight_in HP then weight_out MP;
        ifw %0.4065 queue_weight_in VHP then weight_out HP;
      end;
```

3.8 Comentários

Apresentamos, neste capítulo, a metodologia para construir um controlador para provisionamento de recursos, com o objetivo de obter melhor QoS em domínio DiffServ, a partir da especificação de políticas administrativas. Inicialmente, apresentamos a arquitetura geral de um nó DiffServ. Em seguida, mostramos a arquitetura e metodologia proposta nesse trabalho. Finalmente, apresentamos a metodologia para construir o controlador de QoS: definição da política de gerenciamento, mapeamento da política em parâmetros do controlador fuzzy, construção do controlador e otimização dos parâmetros do controlador.

A metodologia para especificar um controlador de QoS, aqui apresentada, permite a construção de um controlador otimizado, sem exigir do projetista uma escolha de parâmetros eficiente. Apresentaremos, no próximo capítulo, o detalhamento da implementação do controlador utilizado em nosso protótipo.

Capítulo 4

Implementações do protótipo

A PARTIR da metodologia apresentada no capítulo anterior, podemos descrever, nesse capítulo, a construção do mecanismo para provisionamento de recursos em nosso protótipo. O objetivo é validar a metodologia proposta, desde a especificação das políticas definidas para o domínio até a definição do controlador apropriado para diversas topologia e padrões de tráfego.

A experiência foi criar uma aplicação de Telefonia IP cruzando um domínio DiffServ com tráfegos concorrentes. Escolhemos as classes EF *(Expedited Forwarding)* para o tráfego de voz e BE *(Best Effort)* para tráfego concorrente. A classe EF é ideal para aplicações com restrição de tempo, como é a telefonia IP, pois oferece o menor retardo possível. A classe BE representa uma rede IP (Internet), sem garantias de retardo, usada aqui para concorrer com o tráfego da classe EF.

Inicialmente, apresentamos, na seção 4.1, o ambiente de simulação, isto é, os sistemas e ferramentas utilizadas em nossa experiência. Em seguida, na seção 4.2, mostramos a especificação das políticas utilizadas nesse protótipo, tanto para o escalonador como para o condicionador. Na seção 4.3, apresentamos o mapeamento das políticas em funções de pertinência, e na seção 4.4, o mapeamento das políticas em base de regras. Mostramos na seção 4.5 a implementação do controlador do escalonador. Na seção 4.6, apresentamos o controlador convencional, utilizado para comparar o funcionamento do controlador fuzzy. Finalmente, mostramos na seção 4.7, a implementação do código do controlador no simulador *ns-2*.

4.1 Ferramentas

Para validação do modelo proposto, utilizamos o simulador de redes *Network Simulator - ns*[77]. Adicionalmente, utilizamos a ferramenta *PonderToolkit*, que permitiu especificar as políticas administrativas e verificar a correção dessa especificação e a ocorrência de conflitos. A ferramenta *JFS*[76] foi utilizada para construir o controlador fuzzy, a partir da especificação de políticas, e para realizar a otimização dos parâmetros.

4.1.1 Simulador de redes *ns*

A plataforma de simulação utilizada para validação do modelo foi do Network Simulator (*ns*), versão 2.1b8a[77]. Essa versão do *ns* incorpora recurso de arquitetura DiffServ, originário do modelo proposto por Pieda e Ethridge[78].

O modelo DiffServ do *ns* apresenta as seguintes funcionalidades: PHBs, *Assured Forwarding* (AF), *Expedited Forwarding* (EF), *Class Selector* e *Default* (BE); condicionadores, *TSW 2 Color Marker, TSW 3 Color Marker, Single Rate 3CM, Two Rate 3CM* e *Token Bucket*; gerenciamento ativo de filas, RIO-C *(Coupled)*, RIO-D *(Decoupled)*, WRED e *DropTail*; escalonadores, *Weighted Round Robin* (WRR), *Weighted Interleaved Round Round* (WIRR), *Round Robin* (RR) e *Priority* (PRIO).

Acrescentamos novas funções ao modelo DiffServ, que permitissem a alteração de alguns parâmetros de configuração durante uma simulação, não existentes na versão original. Apresentamos, no apêndice B, uma descrição do funcionamento do *ns* e o modelo de simulação desenvolvido para este trabalho.

O modelo DiffServ utilizado nas simulações usou escalonador WRR, filas *Drop Tail* em ambas as classes e condicionador *Token Bucket* para os fluxos da classe EF (a classe BE não foi condicionada).

4.1.2 Ponder Toolkit

A linguagem Ponder é uma linguagem declarativa e orientada a objetos com o objetivo de especificar políticas, tentando aproximar-se das regras de políticas abstratas definidas administrativamente. A ferramenta *Ponder Toolkit* é constituída por um editor de políticas, um compilador da linguagem, um gerador de objetos em *Java* e uma interface de gerenciamento do repositório. Foi utilizada a versão 1.0.1

do *Ponder Policy Editor* e a versão 0.2.1 do *Ponder Compiler*.

Em nosso exemplo utilizamos apenas as ferramentas de edição e compilação, pois não é objeto desse trabalho implementar uma plataforma de gerenciamento de políticas, que a ferramenta Ponder Toolkit se propõe a oferecer. O mapeamento das políticas em Ponder, para funções de pertinência e base de regras do controlador fuzzy, foi realizada manualmente.

Apresentamos, no apêndice A, uma descrição resumida da linguagem Ponder com os comandos utilizados em nossa especificação.

4.1.3 JFS - Ferramenta Fuzzy

O controlador fuzzy utilizado nesse trabalho foi desenvolvido com a ferramenta *JFS*, de Mortensen[76]. Essa ferramenta oferece um ambiente para desenvolvimento do protótipo (especificação das funções de pertinência, regras de inferência e desfuzificador), além de permitir verificação inicial do modelo especificado. Além disso, dispõe de várias ferramentas de otimização, dentre as quais, o algoritmo de Wang-Mendel, algoritmo genético e redes neurais.

Após o desenvolvimento do controlador, é gerada uma biblioteca em código C, que implementa o controlador, sendo, então, integrada ao simulador *ns*. O código gerado por esse programa atende às nossas expectativas e possibilitou implementar as funções de um controlador fuzzy com código eficiente. Apresentamos, no apêndice C, uma descrição do funcionamento do sistema JFS e a construção do controlador fuzzy desenvolvido para este trabalho.

4.2 Especificação das políticas de gerenciamento

A partir da arquitetura do nó DiffServ, mostrada na seção 3.1, podemos definir dois tipos de controlador: um para configurar o escalonador, presente em todos os nós do domínio DiffServ, e um para configurar o condicionador, presente em todos os nós da borda do domínio. Essa divisão permite separar funcionalmente os controladores e aplicar apenas os necessários de acordo com a posição do nó no domínio, se de núcleo ou de borda.

Apresentamos a seguir a listagem das especificações de políticas em Ponder utilizadas em nosso protótipo. Com o objetivo de aumentar a clareza das especificações, separamos o código em duas listagens: na primeira, mostramos a especificação das políticas do controlador do escalonador, existente em todos os nós do domínio; e na segunda, mostramos a especificação das políticas do controlador do condicionador, existente apenas nos nós de borda, onde há condicionador.

Todas as listagens foram estruturadas com os comandos **group** e **type**, pois são aplicáveis a vários nós, sendo que, em cada um deles, existe uma seqüência de comando de autorização e obrigação. Apresentamos um resumo da sintaxe *Ponder* no apêndice A.

4.2.1 Especificação das políticas do escalonador

Mostramos, no código 4.1, a listagem da especificação das políticas do escalonador. As linhas de 1 a 23 definem os parâmetros da política de provisionamento. Nesse caso, limitamos a variação do provisionamento do escalonador entre 10% e 90% da banda total do canal de saída (linha 4). Definimos também uma taxa máxima de retardo de 100 ms e uma taxa de descarte máxima de 20%. No comando **constraint**, definimos as restrições de variação do escalonador a serem utilizadas ao longo do código. O comando **event** define as condições de disparo de eventos.

Código 4.1: Especificação da política do escalonador

```
1  // Especificacao do Controlador do Escalonador
2  //
3  // Define limites minimos e maximos permitidos para o escalonador
4  const
5     minsched = 0.10;   // Escalonador minimo 10%
6     maxsched = 0.90;   // Escalonador maximo 90%
7     MaxDelay = 100;    // Delay maximo 100 ms
8     MaxDrop = 0.2;    // Descarte maximo 20%
9  
10 // Define condicoes de restricao
11 constraint
12    bwMin = bwShare < minsched;
13    bwMax = bwShare > maxsched;
14    bwIncrease = bwShare < maxsched;
15    bwDecrease = bwShare > minsched;
16 
17 event    // Definicao dos eventos
18    lowdelay = 0.3 * MaxDelay
19    mediumdelay = 0.5 * MaxDelay
20    highdelay = 0.7 * MaxDelay
21    lowdrop = 0.3 * MaxDrop
```

```
22    mediumdrop = 0.5 * MaxDrop
      highdrop = 0.7 * MaxDrop
```

Na linha 28, mostramos a definição do nome do grupo de políticas que será aplicado a um determinado nó do domínio, onde observamos os parâmetros de configuração sujeito e alvo.

```
   // Cria tipo grupo para especificar controlador do escalonador
26 type
      // Parametros de entrada: sujeito s e objeto de controle t
28    group schedulerProvisioning (subject s, target t) {
         inst
```

As linhas 30 a 43 especificam políticas de autorização negativa, proibindo que o escalonador seja configurado com valores menores que a banda mínima ou maior que a banda máxima, disparadas pelas restrições *bwMin* e *bwMax*, definidas na primeira seção. O comando **auth-** impede a execução dos comandos de redução e aumento da banda, nas linha 31 e 38, respectivamente.

```
30       // Nao autoriza  reduzir scheduler se menor que minsched
         auth- schedulerMin {
32          subject s;
            target t;
34          action reduceBW();
            when bwMin;
36       }
         // Nao autoriza  aumentar scheduler se maior que maxsched
38       auth- schedulerMax {
            subject s;
40          target t;
            action increaseBW();
42          when bwMax;
         }
```

As linhas 44 a 57 especificam políticas de autorização positiva, permitindo que o escalonador aumente ou reduza a banda, caso esteja dentro dos limites permitidos.

```
44       // Autoriza aumentar escalonador se menor que maxsched
         auth+ increaseSchedulerAuth {
46          subject s;
            target t;
48          action increaseBW();
            when bwIncrease;
50       }
         // Autoriza reduzir escalonador se maior que minsched
```

```
52    auth+ decreaseSchedulerAuth {
        subject s;
54      target t;
        action decreaseBW();
56      when bwDecrease;
      }
```

As linhas 58 a 71 especificam políticas de obrigação, aumentando a banda do escalonador um ou dois níveis quando o retardo na classe EF for médio ou alto, respectivamente. Observar que os comandos colocados são abstratos, considerando apenas o desejo do projetista, sem detalhar valores. Para indicar a intensidade do comando especificamos uma quantidade de unidades, isto é, o comando de 2 unidades é maior do que o comando de 1 unidade.

```
58    // Obriga aumentar 1 unidade escalonador quando retardo do EF e' medio
        oblig increaseScheduler1 {
60      subject s;
        target t;
62      on EF.mediumdelay;
        do increaseBW(level);
64    }
      // Obriga aumentar 2 unidades escalonador quando retardo do EF e' alto
66    oblig increaseScheduler2 {
        subject s;
68      target t;
        on EF.highdelay;
70      do increaseBW(2*level);
      }
```

As linhas 72 a 86 especificam políticas de obrigação, diminuindo a banda do escalonador um ou dois níveis quando o descarte na classe BE for médio ou alto, respectivamente, mantendo a condição de que o retardo na classe EF seja baixo. Essas políticas executam o sentido inverso das políticas anteriores, mantendo-se a forma abstrata de representação, sem especificar valores.

```
72    // Obriga reduzir 1 unidade escalonador quando retardo do EF e' baixo e descarte BE e' medio
      oblig decreaseScheduler1 {
74      subject s;
        target t;
76      on EF.lowdelay && BE.mediumdrop;
        do decreaseBW(level);
```

```
78       }
         // Obriga reduzir 2 unidades escalonador quando retardo do EF e↯
            ' baixo e descarte BE e' alto
80       oblig decreaseScheduler2 {
            subject s;
82          target t;
            on EF.lowdelay && BE.highdrop;
84          do decreaseBW(2*level);
         }
86    }
```

Finalmente, as linhas de 88 e 89 especificam a instanciação de um nó pertencente ao núcleo do domínio. Nessa especificação, o sujeito *gerente_politicas* receberá as autorizações e a responsabilidade de executar as ações de gerenciamento. O alvo das ações de gerenciamento é o *corenode1*. Essa especificação se repetirá para todos os nós do domínio, inclusive os localizados na borda, repetindo-se então, a linha 89 tantas vezes quantas forem necessárias.

```
   // Instancia todos os nos do dominio
88 inst
      group corenode1 = schedulerProvisioning(gerente_politicas,↯
         corenode1);
```

4.2.2 Especificação das políticas do condicionador

Mostramos, no código 4.2, a listagem da especificação das políticas do condicionador. O condicionador na arquitetura DiffServ está associado a um determinado fluxo de dados, geralmente identificado pelo endereço de origem, código (*codepoint*) e endereço de destino. Essa política é representada por regras de filtragem no classificador do condicionador na borda do domínio. Não exibiremos essa especificação por considerá-la trivial e também para melhorar a clareza da listagem.

As linhas de 1 a 4 definem os parâmetros de configuração, a taxa mínima do Token Bucket e a taxa máxima de descarte. Esses valores definem os limites para referência de funcionamento do controlador. Definimos uma taxa mínima do Token Bucket de 50% da taxa nominal e uma taxa de descarte máxima de 10%.

Código 4.2: Especificação da política do condicionador

```
  // Define taxa minima do condicionador em relacao a maxima
2 const
     MinRate = 0.5; // Taxa minima do TB e' 50%
```

```
4   MaxDrop = 0.1; // Taxa perda maxima e' 10%
```

O comando **event**, mostrado nas linhas 5 a 9, especifica os parâmetros para disparo dos eventos que determinam uma ação. Em nosso exemplo, esse valor seria a definição da função de pertinência do controlador fuzzy, especificando o valor $0.3 * MaxDrop$ como média do valor semântico *lowdrop*. Esses parâmetros são globais para toda a especificação.

```
  // Especificacao dos eventos
6 event
    lowdrop = 0.3 * MaxDrop
8   mediumdrop = 0.5 * MaxDrop
    highdrop = 0.7 * MaxDrop
```

Na linha 11, temos a definição do nome do grupo de políticas do condicionador, que será aplicado a um determinado nó de borda do domínio, onde observamos os parâmetros de configuração sujeito, alvo e SLA. Chamamos de SLA a taxa do condicionador contratada, que também será a taxa máxima aplicada.

```
10 type
     group conditionerController (subject s, target t, int sla) {
```

Nas linhas 13 a 22, temos a definição de constantes e restrições. O comando **const** especifica os valores mínimo e máximo da taxa do Token Bucket, baseado no valor do SLA. No comando **constraint** definimos as restrições de variação da taxa do Token Bucket a serem utilizadas ao longo do código.

```
        const
14          minTBrate = MinRate * SLA;
            maxTBrate = SLA ;
16
        // Define condicoes de restricao
18      constraint
            TBMin = TBrate < minTBrate;
20          TBMax = TBrate > maxTBrate;
            TBrateDecrease = TBrate > minTBrate;
22          TBrateIncrease = TBrate < maxTBrate;
```

As linhas 24 a 38 especificam políticas de autorização negativa, proibindo que a taxa do condicionador seja configurada com valores menores que a taxa mínima ou maiores que a taxa máxima. As restrições foram definidas através das variáveis *TBMin* e *TBMax*, mostradas anteriormente. O comando **auth-** nega autorização de execução dos comandos de redução e aumento da banda, nas linha 26 e 33, respectivamente.

```
24      inst
        // Nao autoriza  reduzir taxa condicionador se menor que
           minTBrate
26      auth- TBRateMin {
           subject s;
28         target t;
           action reduceTBrate();
30         when TBMin;
        }
32      // Nao autoriza  aumentar taxa condicionador se maior que
           maxTBrate
        auth- TBRateMax {
34         subject s;
           target t;
36         action increaseTBrate();
           when TBMax;
38      }
```

As linhas 40 a 52 especificam políticas de autorização positiva, permitindo que o controlador aumente ou reduza a taxa do condicionador, caso esteja dentro dos limites permitidos.

```
        // Autoriza aumentar condicionador se menor que maxTBrate
40      auth+  increaseTBRateAuth {
           subject s;
42         target t;
           action increaseTBrate();
44         when TBrateIncrease;
        }
46      // Autoriza reduzir condicionador se maior que maxTBrate
        auth+  decreaseTBRateAuth {
48         subject s;
           target t;
50         action decreaseTBrate();
           when TBrateDecrease;
52      }
```

As linhas 54 a 66 especificam políticas de obrigação de ação, diminuindo a taxa do condicionador um nível ou dois níveis quando o retardo na classe EF em nós do núcleo for médio ou alto, respectivamente. Observar que os comandos são especificados de forma abstrata.

```
        // Diminui 1 unidade do condicionador quando retardo do EF no
           nucleo e' medio
```

```
54      oblig decreaseTBrate1 {
           subject s;
56         target t;
           on EFnucleo.mediumdelay;
58         do decreaseTBrate(level);
        }
60      // Diminui 2 unidades do condicionador quando retardo do EF no nucleo e' alto
        oblig decreaseTBrate2 {
62         subject s;
           target t;
64         on EFnucleo.highdelay;
           do decreaseTBrate(2*level);
66      }
```

As linhas 68 a 81 especificam políticas de obrigação, aumentando a taxa do condicionador. Esse aumento pode ser de um ou dois níveis quando o descarte na classe EF, na borda, for médio ou alto, respectivamente, desde que seja mantida a condição de que o retardo na classe EF, no núcleo, seja baixo. Essas políticas executam o sentido inverso das políticas anteriores, mantendo-se a representação de forma abstrata.

```
        // Aumenta 1 unidade do condicionador quando retardo do EF no nucleo e' baixo e descarte EF e' medio
68      oblig increaseTBrate1 {
           subject s;
70         target t;
           on EFnucleo.lowdelay && EF.mediumdrop;
72         do increaseTBrate(level);
        }
74      // Aumenta 2 unidades do condicionador quando retardo do EF no nucleo e' baixo e descarte EF e' medio
        oblig increaseTBrate2 {
76         subject s;
           target t;
78         on EFnucleo.lowdelay && EF.highdrop;
           do increaseTBrate(2*level);
80      }
     }
```

Finalmente, as linhas 84 e 85 especificam a instanciação de um nó pertencente à borda do domínio. Nessa especificação, o sujeito *gerente_politicas* receberá as autorizações e a responsabilidade de executar as ações de gerenciamento. O alvo das ações de gerenciamento é o *edgenode1*. O parâmetro *SLA* informa a taxa contratada para um determinado usuário para uma determinada classe de serviços. Essa especificação se repetirá para todos os nós de borda do domínio, ou seja, onde existem condicionadores.

```
// Instancia todos os nos do dominio
inst
    group edgenode1 = conditionerController(gerente_politicas,
        edgenode1,sla);
```

4.3 Mapeamento das políticas em funções de pertinência

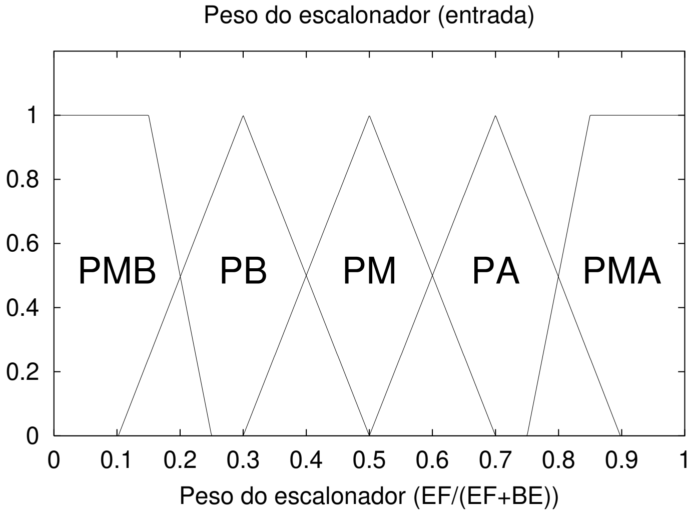
Para possibilitar o uso real, as políticas abstratas precisam ser mapeadas em funções de pertinência do controlador fuzzy, seguindo metodologia apresentada na seção 3.4. O controlador proposto utiliza variáveis lingüísticas triangulares e trapezoidais, pela possibilidade de serem implementadas com código mais simples e eficiente. Realizamos também experiência com uma função gaussiana, porém os resultados obtidos não justificaram a complexidade adicionada. O controlador do escalonador e do condicionador são apresentados a seguir.

4.3.1 Controlador do escalonador

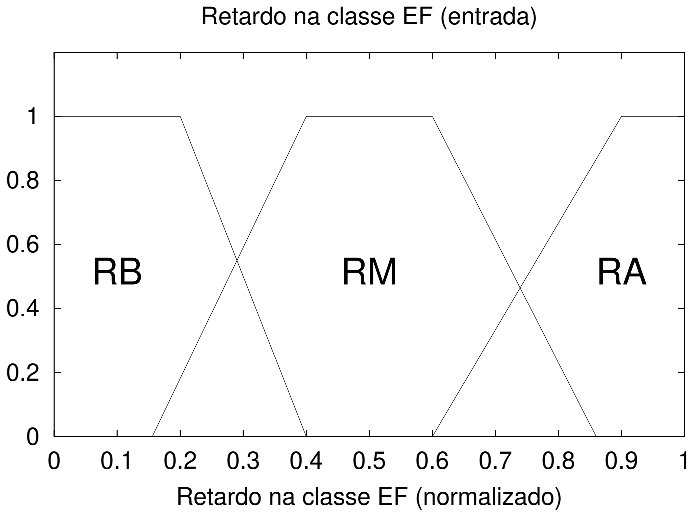
A primeira função de pertinência de entrada é Peso EF/BE, mostrada na figura 4.1(a), representando o peso do escalonador WRR da fila EF, em relação ao peso total (EF+BE). As variáveis semânticas são: "PMB", prioridade muito baixa; "PB", prioridade baixa; "PM", prioridade média; "PA", prioridade alta; e "PMA", prioridade muito alta. Como a relação peso da fila EF por peso total do WRR é sempre um número entre 0 e 1, não foi necessário efetuar a normalização dos valores.

Considerando que o valor de retardo é a métrica principal para avaliação de QoS, nossa segunda função de entrada é o retardo da fila EF, mostrada na figura 4.1(b). O sensor lê o valor de retardo de um pacote durante sua passagem pelo nó, calcula a média das medidas de retardo desde a última operação do controlador e aplica a entrada do controlador. As variáveis semânticas são: "RB", retardo baixo; "RM", retardo médio; e "RA", retardo alto. O valor lido é o tempo em segundos e a variável precisa ser normalizada antes de entrar no controlador.

A terceira função de pertinência de entrada é o percentual de descarte de pacotes na fila BE, mostrada na figura 4.2. O sensor calcula a taxa de pacotes descartados na classe BE, em um determinado intervalo de tempo, com o objetivo de avaliar a possibilidade de redução na prioridade da fila EF. O valor é normalizado em relação ao percentual máximo de perdas. As variáveis semânticas são: "DB", descarte baixo; "DM", descarte médio; e "DA", descarte alto.

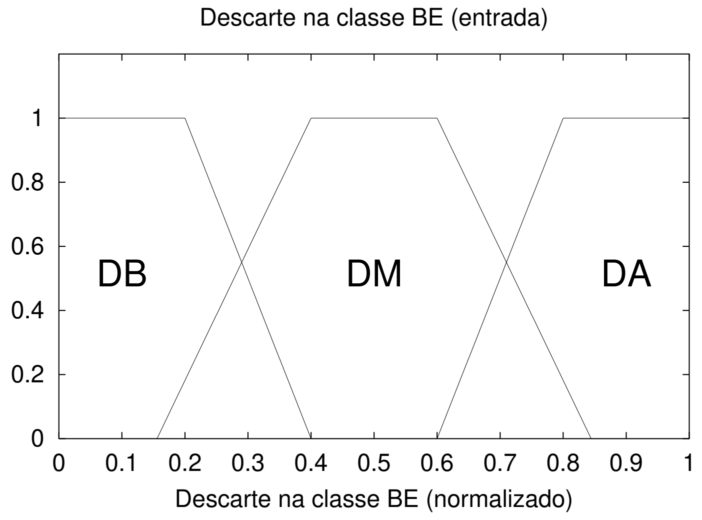


(a) Entrada: peso EF/BE

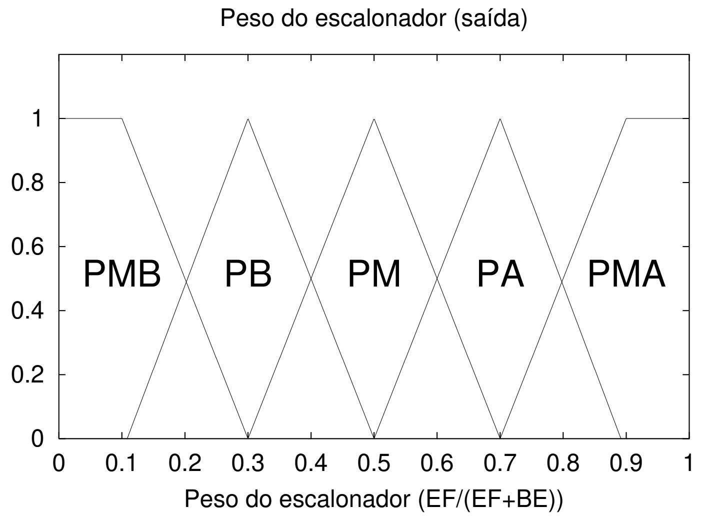


(b) Entrada: retardo EF

Figura 4.1: Funções de pertinência do controlador do escalonador



(a) Entrada: descarte BE

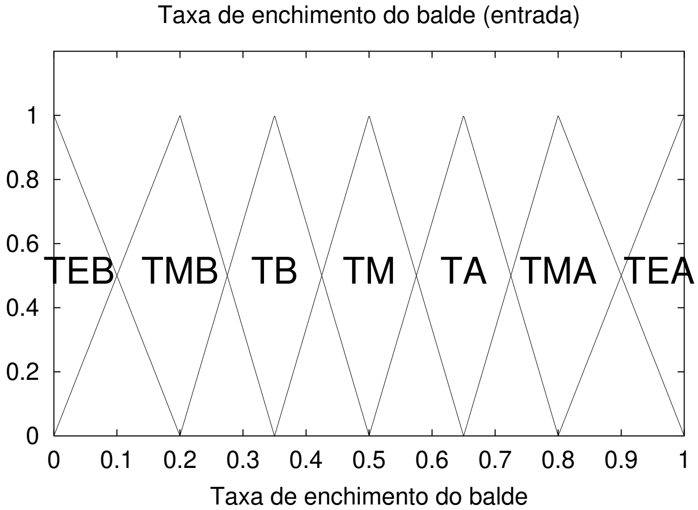


(b) Saída:peso EF/BE

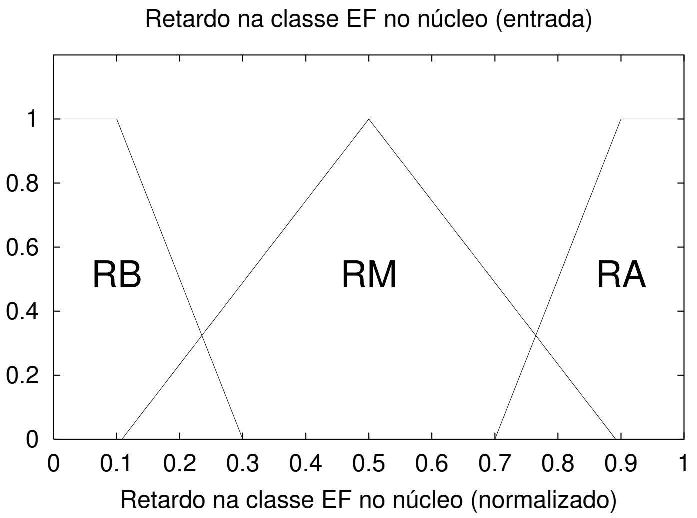
Figura 4.2: Funções de pertinência do controlador do escalonador

4.3.2 Controlador do condicionador

A primeira função de pertinência de entrada do controlador do condicionador é a taxa de enchimento do balde, mostrada na figura 4.3(a). O sensor lê o valor da taxa atual de preenchimento do balde do condicionador, que é normalizado em relação à taxa máxima de enchimento do balde. As variáveis semânticas são: "TEB", taxa extra baixa; "TMB", taxa muito baixa; "TB", taxa baixa; "TM", taxa média; "TA", taxa alta; "TMA", taxa muito alta; e "TEA", taxa extra alta.



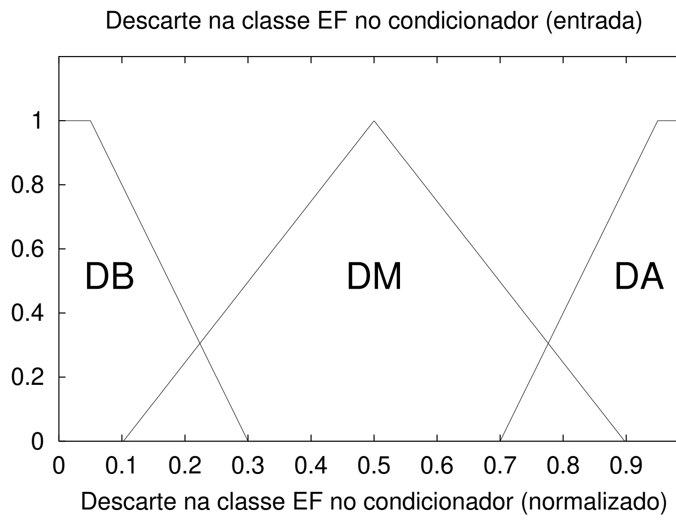
(a) Entrada: taxa enchimento do balde



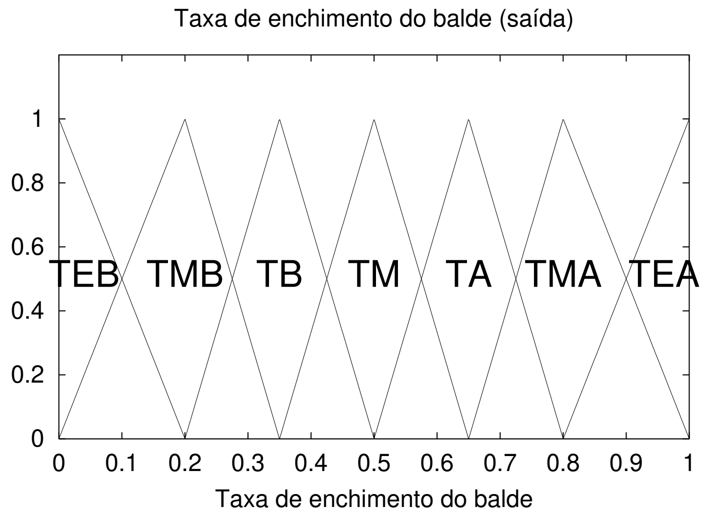
(b) Entrada: retardo EF no núcleo

Figura 4.3: Funções de pertinência do controlador do condicionador

A segunda função de pertinência de entrada é o retardo da classe EF no núcleo do domínio, mostrada na figura 4.3(b). O sensor lê o valor médio de retardo dos pacotes durante a passagem por um nó no núcleo do domínio. Quando o retardo no núcleo começa a aumentar, o gerenciador sinaliza aos nós de borda para reduzir a taxa dos condicionadores, antes de provocar um descarte no núcleo. As variáveis semânticas são: "RB", retardo baixo; "RM", retardo médio; e "RA", retardo alto. O valor lido é o tempo em milisegundos e a variável precisa ser normalizada antes de entrar no controlador.



(a) Entrada: descarte EF no condicionador



(b) Saída: Taxa de enchimento do balde

Figura 4.4: Funções de pertinência do controlador do condicionador

A terceira função de pertinência de entrada é o descarte de pacotes na fila EF, mostrada na figura 4.4(a). Esse sensor mede a quantidade de pacotes descartados no condicionador. O descarte de um pacote indica que a taxa de enchimento do balde é menor que o tráfego entrante no domínio. Retardo baixo no núcleo pode sinalizar um aumento da taxa de entrada no condicionador. O valor é normalizado em relação ao percentual máximo de perdas admitido para a classe EF. As variáveis semânticas são: "DB", descarte baixo; "DM", descarte médio; e "DA", descarte alto.

4.3.3 Funções de pertinência de saída e defuzificador

As funções de saída também foram definidas como funções trapezoidais, pelos motivos expostos anteriormente. Utilizamos o método de desfuzificação de centro de gravidade, pois oferece um bom resultado para esta aplicação[79, 80]. A primeira função de pertinência de saída é o Peso EF/BE, mostrada na figura 4.2(b). Fornece o valor do peso do escalonador WRR da fila EF, em relação ao peso total (EF+BE). As variáveis semânticas são: "PMB", prioridade muito baixa; "PB", prioridade baixa; "PM", prioridade média; "PA", prioridade alta; e "PMA", prioridade muito alta.

A segunda função de pertinência de saída é a taxa de enchimento do balde, mostrada na figura 4.4(b). Fornece o valor da taxa de enchimento do balde do condicionador da classe EF. O resultado é normalizado em relação à taxa máxima. As variáveis semânticas são: "TEB", taxa extra baixa; "TMB", taxa muito baixa; "TB", taxa baixa; "TM", taxa média; "TA", taxa alta; "TMA", taxa muito alta; e "TEA", taxa extra alta.

4.4 Mapeamento das políticas em base de regras

A base de regras é o conjunto de regras SE-ENTÃO dos valores lingüísticos, que representam o comportamento desejado do sistema fuzzy. Conforme dito anteriormente, a base de regras pode ser definida de acordo com a política administrativa. Para nosso exemplo, utilizamos um conjunto de regras priorizando a classe EF, em qualquer situação, e deixando para a classe BE um mínimo de 10% da banda.

A escolha dos operadores fuzzy seguiu as recomendações e metodologia de Cordón e Herrera [80], conforme o tipo de aplicação utilizada em nosso estudo. Os operadores selecionados foram **máximo** para união e interseção (envoltória externa de todas as variáveis semânticas válidas), enquanto a implicação utilizada foi do tipo **mínimo**, a disjunção (OU) do tipo **máximo** e conjunção (E), do tipo **mínimo**. O modelo matemático está detalhado na seção 2.6.2.

O controlador fuzzy, que se estrutura em uma base de regras, pode apresentar problemas de funcionamento quando uma regra errada é incluída em sua base. Para definir as regras do controlador, evitando a ocorrência de erros, utilizamos o algoritmo de Wang-Mendel[71], detalhado na seção 3.5.3. Com essa metodologia conseguimos construir uma base de regras a partir de uma base de conhecimento do comportamento desejado, eliminando eventuais incoerências e erros.

4.4.1 Controlador do escalonador

O controlador do escalonador foi definido através de 22 regras, sintetizadas a seguir, no algoritmo 4.2.

Algoritmo 4.2 Base de regras do controlador do escalonador

Entrada: Peso do escalonador atual, retardo atual da fila EF, descarte na fila BE
Saída: Peso do escalonador
se retardo fila EF é médio (RM) **então**
  prioridade da fila EF é incrementada em 1
fim se
se retardo fila EF é alto (RA) **então**
  prioridade da fila EF é incrementada em 2
fim se
se retardo fila EF é baixo (RB) e descarte fila BE é médio (DM) **então**
  prioridade da fila EF é reduzida em 1
fim se
se retardo fila EF é baixo (RB) e descarte fila BE é alto (DA) **então**
  prioridade da fila EF é reduzida em 2
fim se
FIM

4.4.2 Controlador do condicionador

O controlador do condicionador foi definido através de 29 regras, sintetizadas a seguir, no algoritmo 4.3.

Algoritmo 4.3 Base de regras do controlador do condicionador

Entrada: Taxa do condicionador, retardo da fila EF no núcleo do domínio, descarte na fila EF no condicionador
Saída: Taxa do condicionador
se retardo fila EF no núcleo é médio (RM) **então**
  Taxa do condicionador é reduzida em 1
fim se
se retardo fila EF no núcleo é alto (RA) **então**
  Taxa do condicionador é reduzida em 2
fim se
se retardo fila EF no núcleo é baixo (RB) e descarte na fila EF é médio (DM) **então**
  Taxa do condicionador é aumentada em 1
fim se
se retardo fila EF no núcleo é baixo (RB) e descarte na fila EF é alto (DA) **então**
  Taxa do condicionador é aumentada em 2
fim se
FIM

4.5 Controlador fuzzy do escalonador

Apresentamos, nessa seção, o código JFS do controlador do escalonador que inclui a definição das funções de pertinência e base de regras, antes e após as etapas de otimização. O código JFS do condicionador é apresentado no apêndice C.

4.5.1 Código inicial do controlador e base de regras

A primeira etapa de desenvolvimento do controlador fuzzy é a definição das funções de pertinência. Mostramos, no código 4.3, o programa-jfs inicial do controlador do escalonador.

Código 4.3: Código do controlador do escalonador inicial

```
title "Scheduler Controller";

operators
    and type min;
    or type max;
    imp type min;
```

```
    bunion type max;

domains
    relative_weight type float "EF/BE" 0.0 1.0;
    delay type float "msec" 0.0 1.0;
    drops type float "pkts/s" 0.0 1.0;

input
    queue_weight_in "Queue weight IN" domain relative_weight;
    ef_delay "EF Delay" domain delay;
    be_discard "BE Drops" domain drops;

output
    weight_out "Queue Weight OUT" domain relative_weight defuz centroid;

adjectives
    queue_weight_in VLP 0:1 %0.15:1 %0.25:0;
    queue_weight_in LP center %0.3 base %0.3;
    queue_weight_in MP center %0.5 base %0.3;
    queue_weight_in HP center %0.7 base %0.3;
    queue_weight_in VHP %0.75:0 %0.85:1 1.0:1;

    ef_delay LowDelay 0:1 %0.2:1 %0.4:0;
    ef_delay MediumDelay trapez %0.4 %0.6 base %0.6;
    ef_delay HighDelay %0.6:0 %0.9:1 1.0:1;

    be_discard LowDiscard 0:1 %0.2:1 %0.4:0;
    be_discard MediumDiscard trapez %0.4 %0.6 base %0.6;
    be_discard HighDiscard %0.6:0 %0.8:1 1.0:1;

    weight_out VLP 0:1 0.1:1 0.3:0;
    weight_out LP center %0.3 base %0.3;
    weight_out MP center %0.5 base %0.3;
    weight_out HP center %0.7 base %0.3;
    weight_out VHP 0.7:0 0.9:1 1.0:1;

program
extern jfrd input queue_weight_in ef_delay be_discard
            output weight_out;
```

Podemos notar que, nesse exemplo, não há base de regras, mas apenas a referência para a utilização do algoritmo Wang-Mendel na última linha. O arquivo de dados, usado como referência para criação das regras, é mostrado no código 4.4.

Código 4.4: Arquivo com base de dados para otimização do controlador do escalonador

```
# queue_in  ef_delay      be_drop         queue_out
VLP         LowDelay      LowDiscard      VLP
LP          LowDelay      LowDiscard      LP
MP          LowDelay      LowDiscard      MP
HP          LowDelay      LowDiscard      HP
VHP         LowDelay      LowDiscard      VHP

VLP         LowDelay      MediumDiscard   VLP
LP          LowDelay      MediumDiscard   VLP
MP          LowDelay      MediumDiscard   LP
HP          LowDelay      MediumDiscard   MP
VHP         LowDelay      MediumDiscard   HP

VLP         LowDelay      HighDiscard     VLP
LP          LowDelay      HighDiscard     VLP
MP          LowDelay      HighDiscard     VLP
HP          LowDelay      HighDiscard     LP
VHP         LowDelay      HighDiscard     MP

VLP         MediumDelay   LowDiscard      MP
LP          MediumDelay   LowDiscard      HP
MP          MediumDelay   LowDiscard      VHP
HP          MediumDelay   LowDiscard      VHP
VHP         MediumDelay   LowDiscard      VHP

VLP         MediumDelay   MediumDiscard   LP
LP          MediumDelay   MediumDiscard   MP
MP          MediumDelay   MediumDiscard   HP
HP          MediumDelay   MediumDiscard   VHP
VHP         MediumDelay   MediumDiscard   VHP

VLP         MediumDelay   HighDiscard     LP
LP          MediumDelay   HighDiscard     MP
MP          MediumDelay   HighDiscard     HP
HP          MediumDelay   HighDiscard     VHP
VHP         MediumDelay   HighDiscard     VHP

VLP         HighDelay     LowDiscard      MP
LP          HighDelay     LowDiscard      HP
MP          HighDelay     LowDiscard      VHP
HP          HighDelay     LowDiscard      VHP
VHP         HighDelay     LowDiscard      VHP
```

```
VLP     HighDelay     MediumDiscard   MP
LP      HighDelay     MediumDiscard   HP
MP      HighDelay     MediumDiscard   VHP
HP      HighDelay     MediumDiscard   VHP
VHP     HighDelay     MediumDiscard   VHP

VLP     HighDelay     HighDiscard     MP
LP      HighDelay     HighDiscard     HP
MP      HighDelay     HighDiscard     VHP
HP      HighDelay     HighDiscard     VHP
VHP     HighDelay     HighDiscard     VHP
```

4.5.2 Código do controlador verificado com Wang-Mendel

Após a verificação das regras com o algoritmo de Wang-Mendel, obtemos a listagem apresentada no código 4.5.

Código 4.5: Código do controlador com regras verificadas por Wang-Mendel

```
title "Scheduler Controller";

operators
    and type min;
    or type max;
    imp type min;
    bunion type max;

domains
    relative_weight "EF/BE" 0 1;
    delay "msec" 0 1;
    drops "pkts/s" 0 1;

input
    queue_weight_in "Queue weight IN" relative_weight;
    ef_delay "EF Delay" delay;
    be_discard "BE Drops" drops;

output
    weight_out "Queue Weight OUT" relative_weight;

adjectives
    queue_weight_in VLP 0:1 %0.15:1 %0.25:0;
    queue_weight_in LP %0.3 base %0.3;
    queue_weight_in MP %0.5 base %0.3;
```

```
    queue_weight_in HP %0.7 base %0.3;
    queue_weight_in VHP %0.75:0 %0.85:1 1:1;

    ef_delay LowDelay 0:1 %0.2:1 %0.4:0;
    ef_delay MediumDelay base %0.6 trapez %0.4 %0.6;
    ef_delay HighDelay %0.6:0 %0.9:1 1:1;

    be_discard LowDiscard 0:1 %0.2:1 %0.4:0;
    be_discard MediumDiscard base %0.6 trapez %0.4 %0.6;
    be_discard HighDiscard %0.6:0 %0.8:1 1:1;

    weight_out VLP 0:1 0.1:1 0.3:0;
    weight_out LP %0.3 base %0.3;
    weight_out MP %0.5 base %0.3;
    weight_out HP %0.7 base %0.3;
    weight_out VHP 0.7:0 0.9:1 1:1;

program
    switch;
    case ef_delay LowDelay;
      switch;
      case be_discard LowDiscard;
        ifw %0.7 queue_weight_in LP then weight_out LP;
        ifw %0.7 queue_weight_in MP then weight_out MP;
        ifw %0.7 queue_weight_in HP then weight_out HP;
      end;
      switch;
      case be_discard MediumDiscard;
        ifw %0.7 queue_weight_in MP then weight_out LP;
        ifw %0.7 queue_weight_in HP then weight_out MP;
        ifw %0.7 queue_weight_in VHP then weight_out HP;
      end;
      switch;
      case be_discard HighDiscard;
        ifw %0.7 queue_weight_in between VLP and MP then weight_out ↯
           VLP;
        ifw %0.7 queue_weight_in HP then weight_out LP;
        ifw %0.7 queue_weight_in VHP then weight_out MP;
      end;
      ifw %0.7 queue_weight_in VLP then weight_out VLP;
      ifw %0.7 queue_weight_in between VLP and LP and be_discard ↯
         between
        MediumDiscard and HighDiscard then weight_out VLP;
    end;
    switch;
    case ef_delay between MediumDelay and HighDelay;
```

```
      switch;
      case be_discard LowDiscard;
        ifw %0.7 queue_weight_in VLP then weight_out MP;
        ifw %0.7 queue_weight_in LP then weight_out HP;
        ifw %0.7 queue_weight_in between MP and VHP then weight_out VHP;
      end;
      ifw %0.7 queue_weight_in between HP and VHP then weight_out VHP;
    end;
    switch;
    case ef_delay MediumDelay;
      switch;
      case be_discard between MediumDiscard and HighDiscard;
        ifw %0.7 queue_weight_in VLP then weight_out LP;
        ifw %0.7 queue_weight_in LP then weight_out MP;
        ifw %0.7 queue_weight_in MP then weight_out HP;
      end;
    end;
    switch;
    case ef_delay HighDelay;
      ifw %0.7 queue_weight_in VLP then weight_out MP;
      ifw %0.7 queue_weight_in LP then weight_out HP;
      ifw %0.7 queue_weight_in between MP and VHP then weight_out VHP;
    end;
    ifw %0.7 queue_weight_in VHP and be_discard LowDiscard then weight_out VHP
      ;
```

4.5.3 Código do controlador otimizado com AG

Após a verificação das regras com o algoritmo Wang-Mendel podemos realizar a otimização com algoritmo genético. Podemos notar que, nas definições das funções de pertinência e na base de regras, alguns valores apresentam o caracter % diante das variáveis que desejamos otimizar. Após a otimização obtemos o código 4.6 mostrado a seguir.

Código 4.6: Código do controlador do escalonador otimizado com AG

```
title "Scheduler Controller";

operators
    and type min;
    or type max;
    imp type min;
```

```
    bunion type max;

domains
    relative_weight "EF/BE" 0 1;
    delay "msec" 0 1;
    drops "pkts/s" 0 1;

input
    queue_weight_in "Queue weight IN" relative_weight;
    ef_delay "EF Delay" delay;
    be_discard "BE Drops" drops;

output
    weight_out "Queue Weight OUT" relative_weight;

adjectives
    queue_weight_in VLP 0:1 %0.15:1 %0.2685:0.0026;
    queue_weight_in LP %0.3 base %0.3;
    queue_weight_in MP %0.5 base %0.1814;
    queue_weight_in HP %0.7 base %0.3;
    queue_weight_in VHP %0.75:0 %0.85:0.999 1:1;

    ef_delay LowDelay 0:1 %0.2:1 %0.22:0.0004;
    ef_delay MediumDelay base %0.6 trapez %0.3 %0.5;
    ef_delay HighDelay %0.6:0 %0.7513:1 1:1;

    be_discard LowDiscard 0:1 %0.2:1 %0.4:0.0003;
    be_discard MediumDiscard base %2.0106 trapez %0.3162 %0.6;
    be_discard HighDiscard %0.6:0 %0.8129:0.9999 1:1;

    weight_out VLP 0:1 0.1:1 0.3:0;
    weight_out LP %0.35 base %0.3;
    weight_out MP %0.57 base %0.31;
    weight_out HP %0.76 base %0.3;
    weight_out VHP 0.7:0 0.9:1 1:1;

program
    switch;
    case ef_delay LowDelay;
      switch;
      case be_discard LowDiscard;
        ifw %0.7 queue_weight_in LP then weight_out LP;
        ifw %0.4793 queue_weight_in MP then weight_out MP;
        ifw %0.4152 queue_weight_in HP then weight_out HP;
      end;
      switch;
```

```
    case be_discard MediumDiscard;
      ifw %0.3236 queue_weight_in MP then weight_out LP;
      ifw %0.2917 queue_weight_in HP then weight_out MP;
      ifw %0.4065 queue_weight_in VHP then weight_out HP;
    end;
    switch;
    case be_discard HighDiscard;
      ifw %0.3623 queue_weight_in between VLP and MP then weight_out VLP;
      ifw %0.2814 queue_weight_in HP then weight_out LP;
      ifw %0.4580 queue_weight_in VHP then weight_out MP;
    end;
    ifw %0.3232 queue_weight_in VLP then weight_out VLP;
    ifw %0.2659 queue_weight_in between VLP and LP and be_discard between
      MediumDiscard and HighDiscard then weight_out VLP;
  end;
  switch;
  case ef_delay between MediumDelay and HighDelay;
    switch;
    case be_discard LowDiscard;
      ifw %0.2892 queue_weight_in VLP then weight_out MP;
      ifw %0.2572 queue_weight_in LP then weight_out HP;
      ifw %0.3042 queue_weight_in between MP and VHP then weight_out VHP;
    end;
    ifw %0.2258 queue_weight_in between HP and VHP then weight_out VHP;
  end;
  switch;
  case ef_delay MediumDelay;
    switch;
    case be_discard between MediumDiscard and HighDiscard;
      ifw %0.3029 queue_weight_in VLP then weight_out LP;
      ifw %0.3916 queue_weight_in LP then weight_out MP;
      ifw %0.3505 queue_weight_in MP then weight_out HP;
    end;
  end;
  switch;
  case ef_delay HighDelay;
    ifw %0.2742 queue_weight_in VLP then weight_out MP;
    ifw %0.4039 queue_weight_in LP then weight_out HP;
    ifw %0.0734 queue_weight_in between MP and VHP then weight_out VHP;
  end;
  ifw %0.2758 queue_weight_in VHP and be_discard LowDiscard then
```

```
    weight_out
  VHP;
```

Com o código otimizado podemos convertê-lo para código em linguagem "C" que será compilado ao simulador de redes *ns-2*.

4.6 Controlador convencional

Com o objetivo de validar nossa proposta, e tendo em vista que os resultados com a utilização do controlador fuzzy seriam melhores que a situação sem controlador, definimos, para comparação, um controlador PD (Proporcional e Derivativo) digital simples, que executa um controle intuitivo, com as características mostradas a seguir, no algoritmo 4.4:

A idéia desse controlador é guardar as últimas três medidas de retardo da classe EF, ajustar uma reta a esses pontos. A inclinação dessa reta é aplicada ao peso do escalonador, aumentando ou reduzindo conforme a inclinação da reta.

Algoritmo 4.4 Algoritmo do controlador convencional

Entrada: Peso do escalonador atual, retardo atual da fila EF, descarte na fila BE
Saída: Peso do escalonador
Calcular média das três últimas medidas de retardo.
se média > valor máximo **então**
  Calcula inclinação da reta ajustada aos 3 pontos
  Aplica valor de inclinação (positiva) ao valor do peso do escalonador
fim se
se média < valor mínimo **então**
  Calcula inclinação da reta ajustada aos 3 pontos
  Aplica valor de inclinação (negativa) ao valor do peso do escalonador
fim se
FIM

4.7 Implementação do código do controlador

Foi desenvolvida uma biblioteca com as classes dos controladores implementados. Foram criados um controlador convencional (ConvCtrl) e um controlador fuzzy (FuzzyCtrl). O diagrama UML (*Unified Modeling Language*) da classe ConvCtrl é apresentado na figura 4.5 e da classe FuzzyCtrl, na figura 4.6. Essas classes realizam o gerenciamento do domínio e executam, por exemplo, o ajuste do escalonador e a redução de tráfego nos nós de borda, quando há retardos grandes no núcleo.

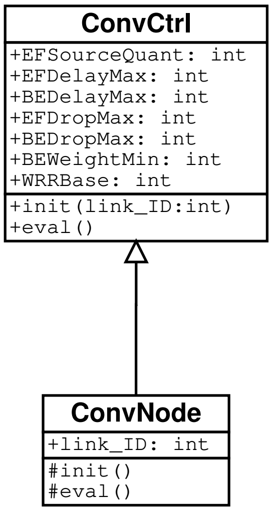


Figura 4.5: Diagrama de classes do controlador convencional

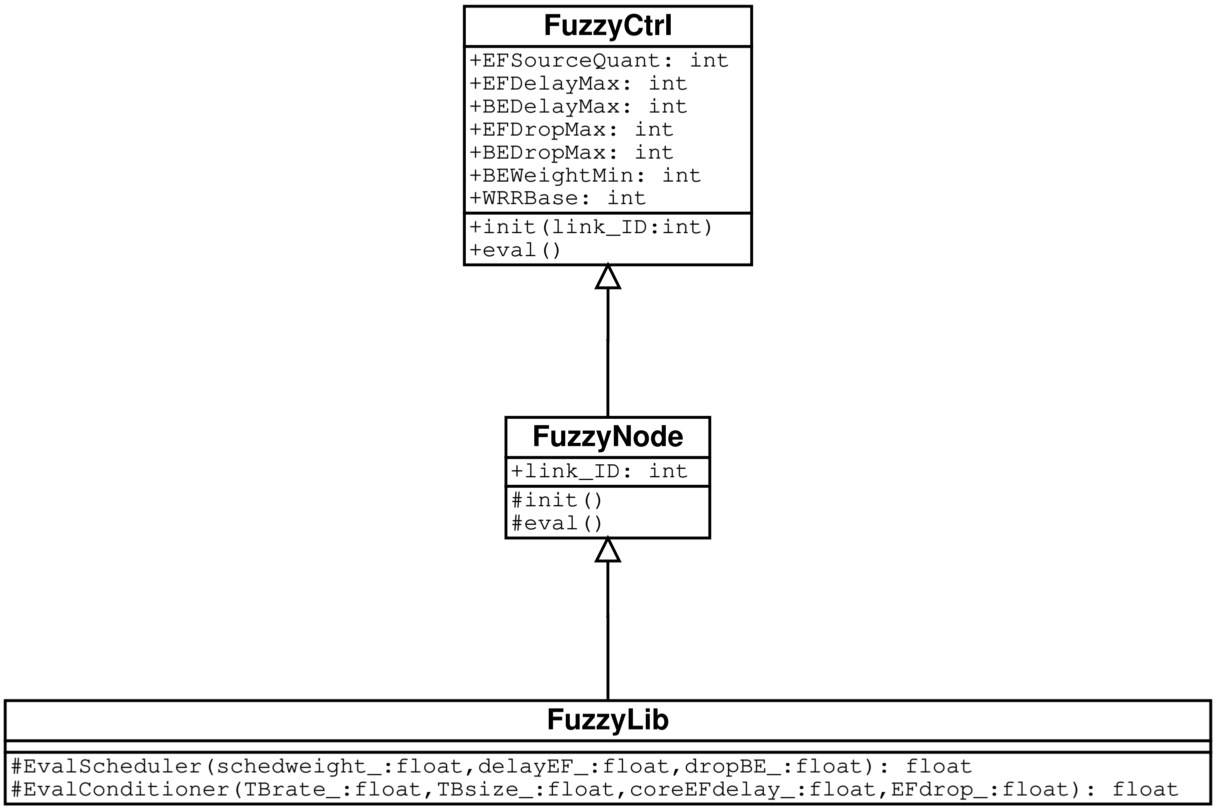


Figura 4.6: Diagrama de classes do controlador fuzzy

O controle do nó é realizado pelas classes ConvNode e FuzzyNode, que lêem as variáveis do nó, processam essas informações e realizam a alteração dos parâmetros de configuração. No caso da classe FuzzyNode, há uma classe FuzzyLib com todas as bibliotecas que implementam o controlador fuzzy. Todas as classes foram escritas em OTcl, enquanto a classe FuzzyLib foi escrita em C++, para melhorar o desempenho e possibilitar um tempo de simulação menor.

4.8 Comentários

Apresentamos, nesse capítulo, a descrição do processo de desenvolvimento do protótipo utilizado para validar a metodologia proposta. Mostramos também o ambiente de simulação utilizado e a ferramenta para desenvolvimento do controlador fuzzy. Em seguida, detalhamos o mapeamento das políticas em parâmetros do controlador fuzzy, suas funções de pertinência e base de regras, e apresentamos o controlador convencional, utilizado para comparação com o controlador fuzzy. Finalmente, apresentamos a implementação do código no simulador de redes.

Após a construção do controlador realizamos a simulação dos diversos cenários apresentados a seguir. São os resultados obtidos com o referido modelo de simulação que apresentaremos no próximo capítulo.

Capítulo 5

Resultados

APRESENTAMOS, nesse capítulo, os resultados das medidas de avaliação de desempenho, quais sejam, o retardo fim-a-fim e o descarte de um fluxo pertencente à classe EF. Para cada medida, apresentamos os gráficos e tabelas relativos ao tráfego CBR e On-off exponencial. As tabelas mostram média e percentil do retardo e taxa de descarte em cada situação. Mostramos os resultados em um domínio DiffServ sem controlador, com controlador convencional e com o controlador fuzzy.

Inicialmente, apresentamos as topologias de simulação, na seção 5.1, e os modelos de tráfego utilizados, na seção 5.2. A seguir, na seção 5.3, mostramos as medidas que foram avaliadas para verificar o funcionamento do controlador proposto. Apresentamos, na seção 5.4, o processo de otimização com algoritmo genético. Finalmente, mostramos, na seção 5.5, os resultados da simulação em uma topologia simples, na seção 5.6, uma análise do comportamento do controlador quando se varia alguns parâmetros de rede, e, na seção 5.7, os resultados da simulação em uma topologia complexa.

5.1 Topologias de simulação

O primeiro exemplo verificado foi de uma topologia simples, mostrada na seção 5.1.1. O objetivo dessa simulação foi verificar o comportamento do controlador em uma topologia simples bem conhecida, para avaliar seu funcionamento sob condições controladas. O segundo exemplo verificado foi de uma topologia complexa, descrita na seção 5.1.2. O objetivo dessa simulação foi avaliar o funcionamento do controlador em uma situação semelhante às encontradas no mundo real.

5.1.1 Avaliação em uma topologia simples

A topologia simples, utilizada para a simulação, é apresentada na figura 5.1. Essa topologia forma um domínio DiffServ com cinco nós, sendo dois de núcleo e três de borda (a quantidade real de fontes não é mostrada na figura). Podemos notar que existem dois pontos de congestionamento, o primeiro entre os nós de borda e do núcleo e outro entre dois nós de núcleo.

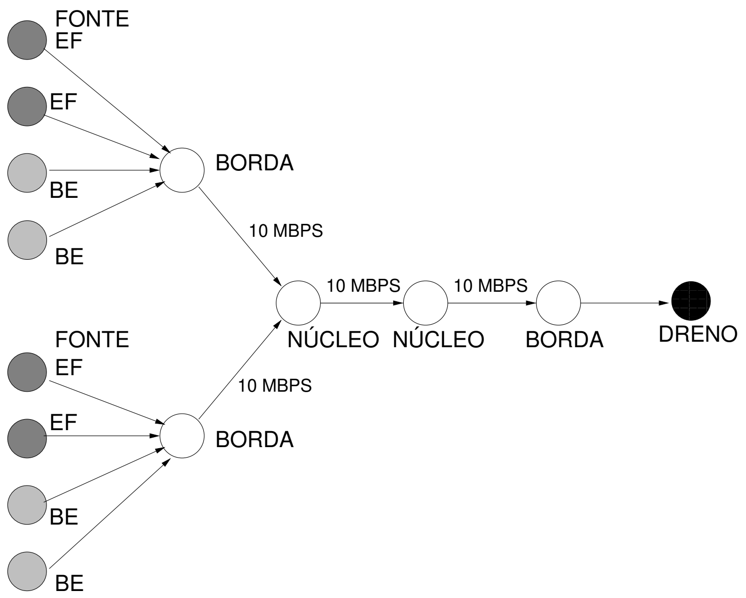


Figura 5.1: Topologia de simulação simples.

5.1.2 Avaliação em uma topologia complexa

O diagrama genérico da topologia complexa utilizada é mostrada na figura 5.2. Trata-se de um domínio DS constituído por 40 nós, sendo 30 de núcleo e 10 de borda. Definimos, na borda, 5 nós como entrada e 5 nós como saída. A topologia foi criada com o pacote *gt-itm*, incluído na distribuição do *ns*[77].

Para validar nossa proposta em um ambiente semelhante ao real, utilizamos três topologias diferentes criadas aleatoriamente, cujas métricas são mostradas na tabela 5.1. Preferimos usar topologias aleatórias no lugar de topologias reais conhecidas porque possibilita a generalização da metodologia proposta.

Zegura et al.[81] mostraram que é possível criar topologias randômicas com métricas das redes reais, porém com o tamanho e a complexidade desejados. Para cada topologia escolhemos um tipo de distribuição para definir a posição dos nós e das

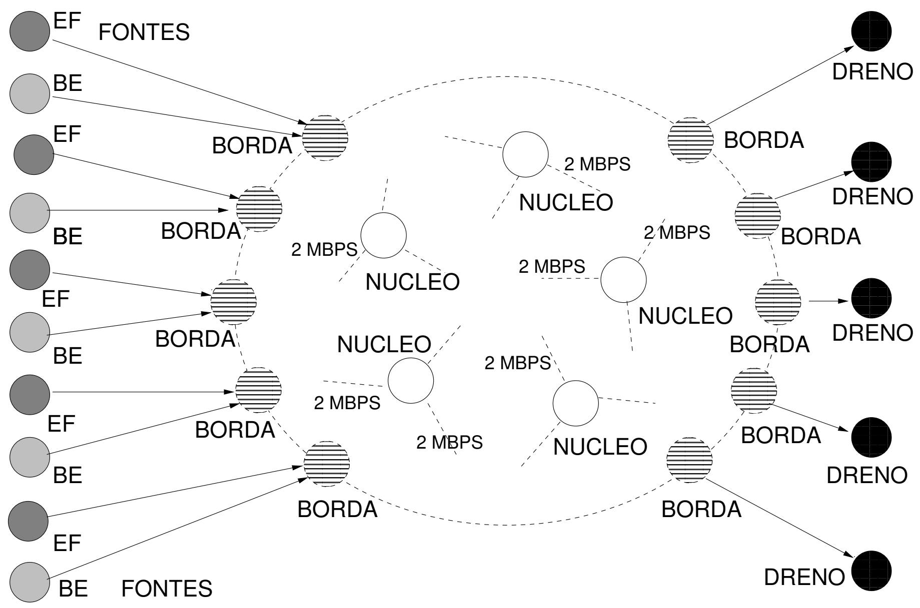


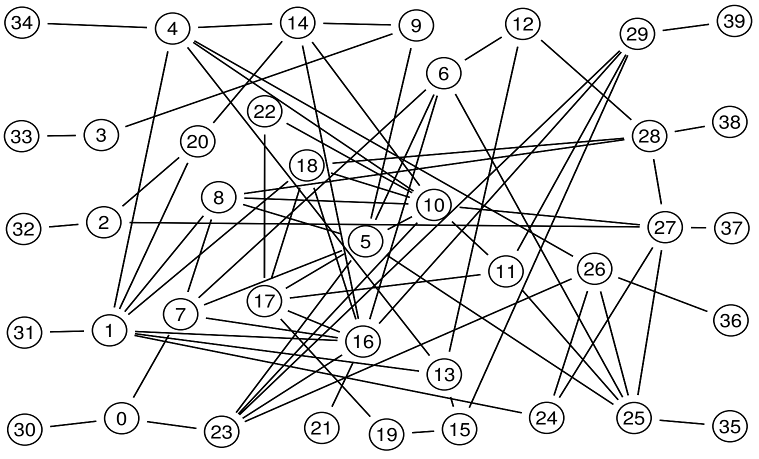
Figura 5.2: Topologia genérica de simulação complexa.

conexões. Todos os tipos de distribuição baseiam-se na seguinte proposição: a probabilidade de dois nós estarem conectados decresce conforme aumenta a distância entre eles. Escolhemos os tipos de distribuição e parâmetros mais apropriados para modelagem da Internet[82, 83].

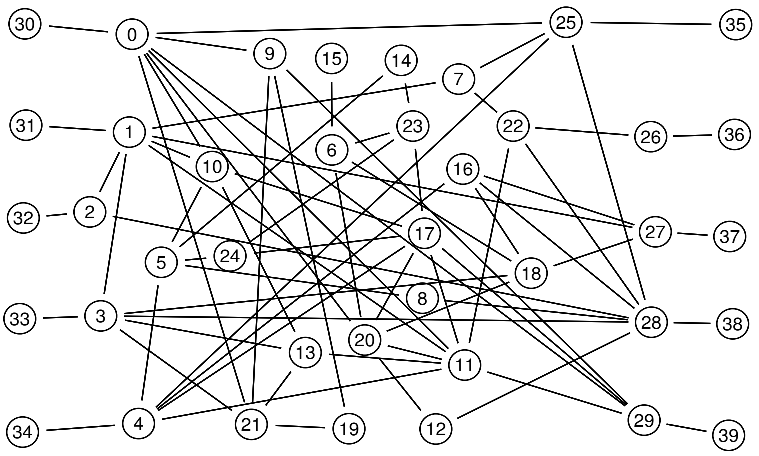
Tabela 5.1: Métricas usadas nas topologias complexas

Topologia	Tipo	Grau Médio (hh, ll, hl)	Diâmetro (hh, ll, hl)	Bicomp
Topologia 1	Waxman	4.333	5, 71, 113	3
Topologia 2	Waxman 2	4.133	5, 105, 147	3
Topologia 3	Exponencial	4.133	6, 95, 102	3

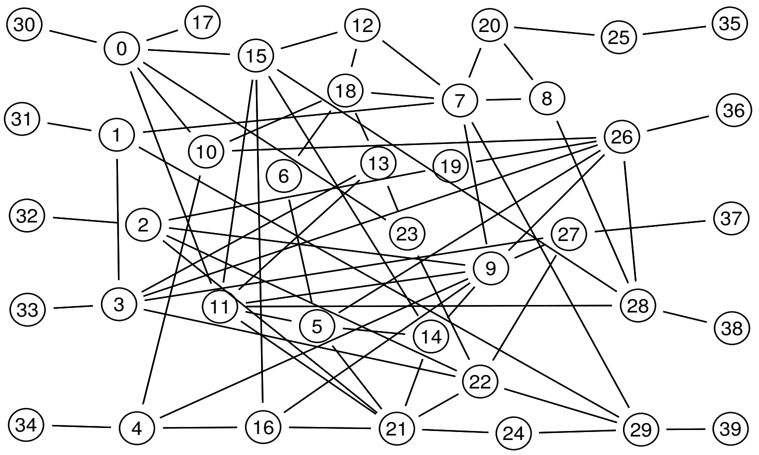
Na tabela 5.1, *Tipo* é o tipo da distribuição, isto é, o algoritmo utilizado para calcular a probabilidade de existir uma conexão entre dois nós. O algoritmo *Waxman*[84] é o modelo de grafo mais comum, onde a probabilidade de existir uma conexão do nó u para o nó v é dado por: $P(u,v) = \alpha\ e^{-d/(\beta L)}$, onde α e β são parâmetros do modelo, d é a distância Euclidiana de u até v e L é a distância máxima entre dois nós. O algoritmo *Waxman 2* é uma variante do modelo Waxman, onde d é substituído por um número randômico entre 0 e L[84]. O algoritmo *Exponencial* é dado por: $P(u,v) = \alpha\ e^{-d/(L-d)}$, onde a probabilidade de existir uma conexão entre dois nós decresce exponencialmente de acordo com a distância entre dois nós.



(a) Topologia 1



(b) Topologia 2



(c) Topologia 3

Figura 5.3: Diagrama das topologias aleatórias reais

A coluna *Grau Médio* é o grau médio de nós, isto é, a relação entre nós e conexões no grafo. *Diâmetro* é o diâmetro do grafo, isto é, a distância do par de nós mais afastados. Três tipos de distância são avaliados: quantidade de saltos (hh), distância Euclideana (ll), distância total (hl), isto é, saltos mais distância. *Bicomp* é o número de componentes biconectados, isto é, a quantidade máxima de conexões entre três nós dentro de um mesmo círculo. Mais detalhes sobre métricas de topologias de redes são mostradas por Zegura[81] e Calvert[83].

A figura 5.3 mostra o diagrama das três topologias utilizadas em nossa simulação, conforme algoritmos e métricas apresentadas na tabela 5.1. Lembramos que a posição real dos nós no plano do domínio poderá não ser a mostrada na figura, com o objetivo de melhorar a visualização da topologia.

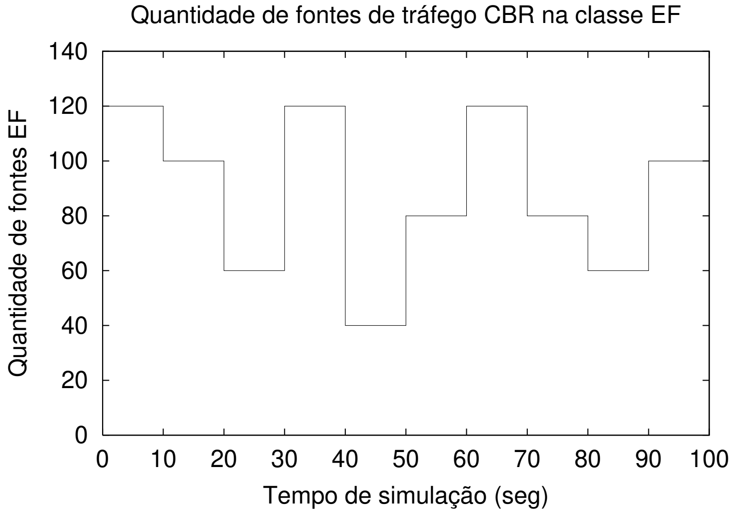
5.2 Modelo de tráfego de simulação

A aplicação de Telefonia IP foi implementada com tráfegos CBR e On-Off exponencial, sobre protocolo UDP. O tráfego CBR tem comportamento determinístico e exige mais banda da rede, enquanto o tráfego On-Off com distribuição exponencial é mais próximo de uma conversação normal. No entanto, a taxa média das fontes On-Off é sensivelmente menor que no caso CBR, por isso adicionamos mais fontes de tráfego On-Off para se obter o congestionamento desejado. O tráfego de voz foi direcionado para classe EF[25] e o tráfego concorrente para classe BE, sendo ambos CBR/UDP.

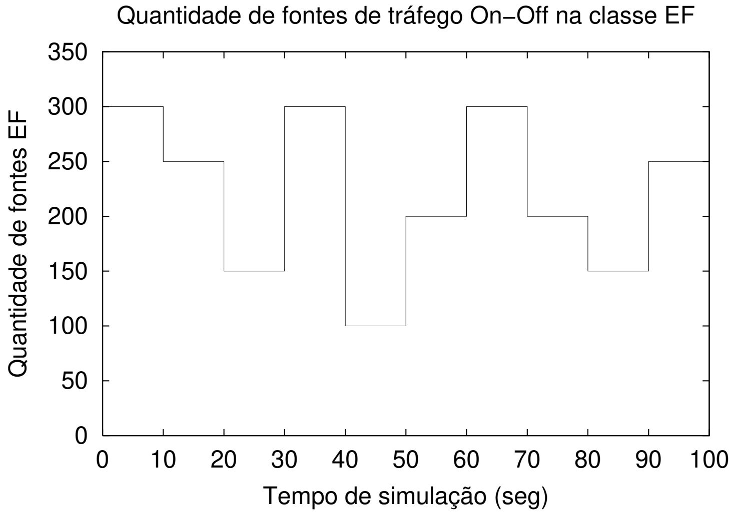
No modelo mostrado na seção 5.2.1, a quantidade de fontes de tráfego foi definida de maneira a se verificar o seu comportamento em uma situação controlada. No segundo caso, mostrado na seção 5.2.2, a quantidade de fontes de tráfego foi definida aleatoriamente, usando parâmetros de tráfego de voz usuais, com o objetivo de fazer essa verificação em uma simulação realista.

5.2.1 Modelo de fonte de tráfego simples

A figura 5.4 mostra a quantidade de fontes de tráfego de voz (classe EF), utilizada durante o período de teste de 100 segundos. A variação de tráfego serviu para demonstrar o funcionamento do controlador nessa situação. Cada fonte de voz CBR foi definida com 64 Kbps, ou seja, um canal de voz PCM. No caso de fonte On-off, utilizamos uma taxa de 64 Kbps, com tempo de rajada de 400 ms e tempo de silêncio de 600 ms, representando uma taxa média de 25,6 Kbps. O tamanho do pacote escolhido foi de 576 bytes, que corresponde a 91,7% dos pacotes de um codificador G.711 (PCM) a 64 Kbps [85].



(a) Fontes CBR



(b) Fontes On-Off Exponencial

Figura 5.4: Quantidade de fontes de tráfego EF durante a simulação

Enquanto o tráfego CBR variou de 40 a 120 fontes, o tráfego On-Off variou de 100 a 300 fontes, exatamente porque a taxa média do tráfego On-Off é aproximadamente 2,5 vezes menor. Utilizamos 300 fontes BE concorrentes, com taxa também de 64 Kbps. O tempo de propagação de cada enlace de 10 Mbps é de 5 ms. Todas as filas têm o tamanho de 50 pacotes, representando um retardo máximo de aproximadamente 23 ms por nó.

5.2.2 Modelo de fonte de tráfego complexa

Para simular uma situação real de conexões de canais de voz, a quantidade de fontes de tráfego varia durante o tempo de simulação de 30 minutos. Com isso, verifica-se o funcionamento do controlador. Essas variações seguem uma distribuição exponencial com os seguintes padrões de tempo de tráfego ativo e tráfego inativo em segundos: (120,60), (60,30), (180,60), (60,20). Novamente, cada fonte CBR foi definida como um canal PCM, com taxa de 64 KBPS. Na fonte On-Off, foi usada taxa de 64 Kbps, com rajada de 400 ms e repouso de 600 ms, resultando em uma taxa média de 25,6 Kbps. O tamanho dos pacotes é de 576 bytes para ambos os casos.

A quantidade de fontes CBR varia de 0 a 150, resultando em uma média de 93 fontes ativas enuanto a quantidade de fontes On-Off varia de 0 a 300 resultando em uma média de 186 fontes ativas durante a simulação. Foram usadas, respectivamente, 150 e 300 fontes CBR 64 Kbps, como tráfego competitivo, na classe BE.

O tempo de propagação de cada canal de 2 Mbps foi definido como 10ms. Todas as filas têm um tamanho máximo de 50 pacotes, produzindo um retardo máximo de 115 ms em cada nó. O modelo de simulação usou escalonador WRR, filas Drop Tail em ambas as classes e condicionador Token Bucket na classe EF, em todos os nós de borda (a classe BE não foi condicionada).

5.3 Medidas avaliadas

Tendo em vista que nosso exemplo se propõe a verificar a qualidade de serviço, as medidas realizadas foram:

- Retardo de um pacote pertencente a um fluxo de voz sobre IP classificado na classe EF.
- Perdas de pacotes pertencentes à classe EF, ao longo do tempo.
- Percentil 50, 90 e 95 do retardo da classe EF.

- Percentil 50, 90 e 95 da variação do retardo (jitter) da classe EF.
- Taxa de perda de pacotes da classe EF.
- Taxa de perda de pacotes da classe BE.

Foram realizadas medidas para fonte de tráfego CBR e On-Off exponencial. Percentil 50 significa que 50% de todos os pacotes têm um retardo e variação de retardo menor que este valor. Percentil 50 também é chamado mediana. Analogamente, percentis 90 e 95 significam retardo ou variação de retardo máximo para 90% e 95% dos pacotes, respectivamente.

A taxa de perda de pacotes significa a quantidade total de pacotes descartados em relação à quantidade total de pacotes transmitidos para cada classe e controlador usado.

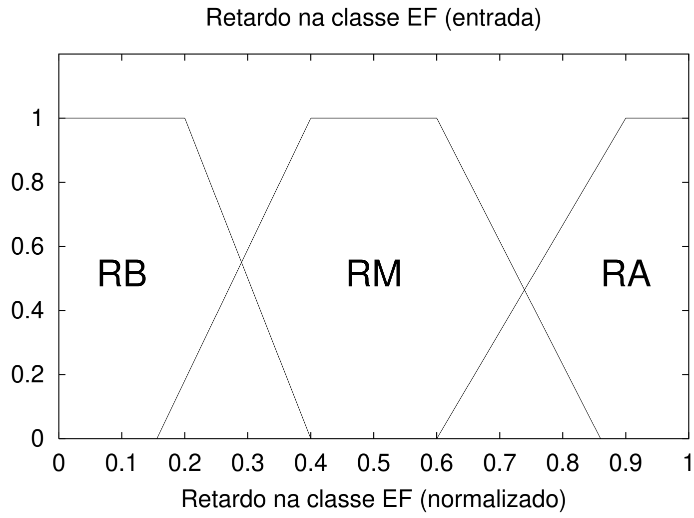
Para avaliar o impacto da alteração dos parâmetros de rede, apresentamos um gráfico com o retardo da classe EF, quando alteramos o tempo de atuação do controlador, e o comportamento do retardo, quando variamos o tamanho dos pacotes transmitidos.

5.4 Resultado da otimização

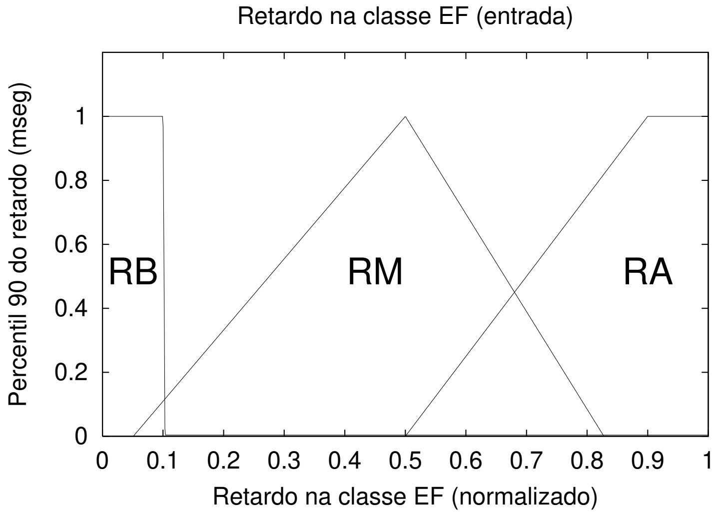
Apresentamos, nessa seção, os resultados da otimização do controlador fuzzy. Para efeito de comparação, mostramos os valores antes e após a otimização. Inicialmente, mostramos a mudança na forma da função de pertinência e, em seguida, a melhoria do retardo fim-a-fim na classe EF após a otimização. Nessa seção, todos os exemplos foram realizados na topologia simples, apresentada na seção 5.1.1, e usando o modelo de tráfego simples determinístico, mostrado na seção 5.2.1.

5.4.1 Funções de pertinência após otimização

Após a realização do procedimento de otimização mostrado na seção 3.7, obtivemos como resultado novas funções de pertinência para o controlador fuzzy. Apresentamos na figura 5.5 a função de pertinência da variável **retardo na classe EF**, utilizada pelo controlador do escalonador. A figura 5.5(a) mostra a função de pertinência original, especificada pelo projetista, onde notamos uma distribuição equilibrada dos valores semânticos ao longo do intervalo de operação.



(a) Controlador não otimizado

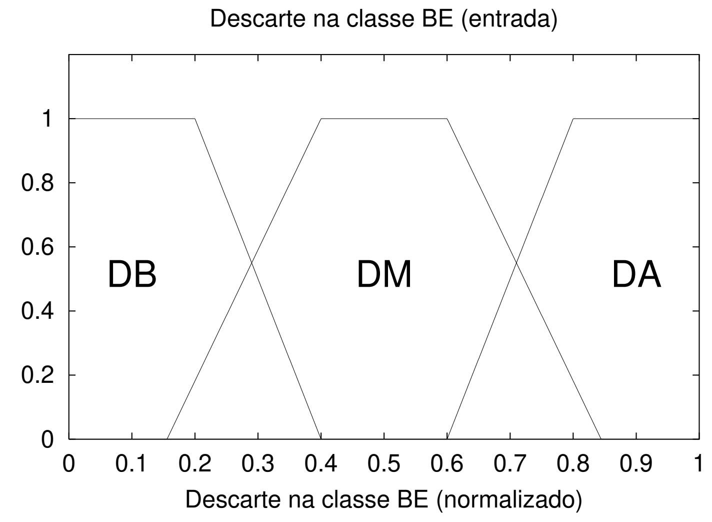


(b) Controlador otimizado

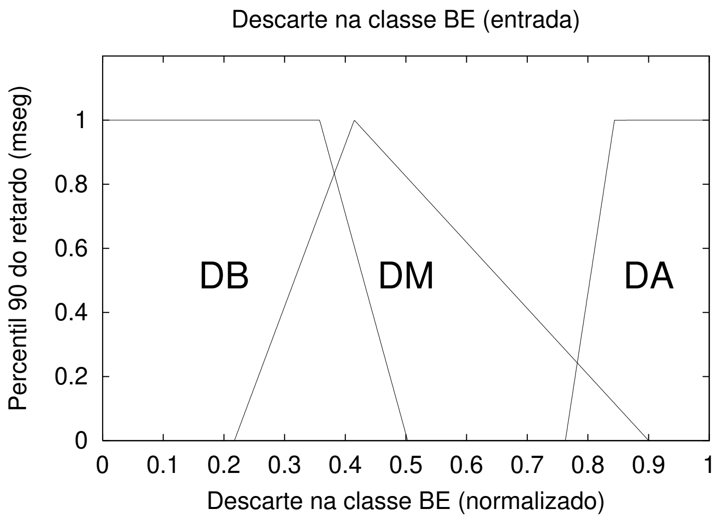
Figura 5.5: Função de pertinência de entrada retardo na classe EF

A figura 5.5(b) mostra a função de pertinência otimizada, obtida após a aplicação da otimização através de algoritmo genético. Podemos notar que o valor semântico "RB"(Retardo Baixo) foi reduzido ao mínimo possível e que o valor semântico "RM"(Retardo Médio) e "RA"(Retardo Alto) sofreram um leve deslocamento para aproximá-los ao valor "RB". Deslocar todos os valores semânticos para a esquerda determina que a base de regra executará mais cedo ações de aumento da banda de saída do escalonador, fazendo que se obtenha um valor de retardo menor, comparando-se com o controlador original não otimizado.

Apresentamos, na figura 5.6, a função de pertinência da variável **descarte na classe BE**, utilizada pelo controlador do escalonador. A figura 5.6(a) mostra a função de pertinência original, especificada pelo projetista, onde, mais uma vez, notamos uma distribuição equilibrada dos valores semânticos ao longo do intervalo de operação.



(a) Controlador não otimizado



(b) Controlador otimizado

Figura 5.6: Função de pertinência de entrada descarte na classe BE

A figura 5.6(b) mostra a função de pertinência otimizada, obtida após a aplicação da otimização através de algoritmo genético. Podemos notar que o valor semântico "DB" (Descarte Baixo) foi aumentado e deslocado para a direita, e que o valor semântico "DA" (Descarte Alto) foi reduzido, sofrendo um deslocamento para a direita. Como o valor semântico "DB" obriga a manter a configuração do escalonador, esse deslocamento para a direita determina que a configuração será mantida por um tempo maior, isto é, que o controlador reagirá mais lentamente ao descarte na classe BE. Além disso, o valor semântico "DA", que determina a reação de aumentar a fatia do escalonador para a classe BE, também foi deslocado para a direita, e portanto será disparado mais tardiamente.

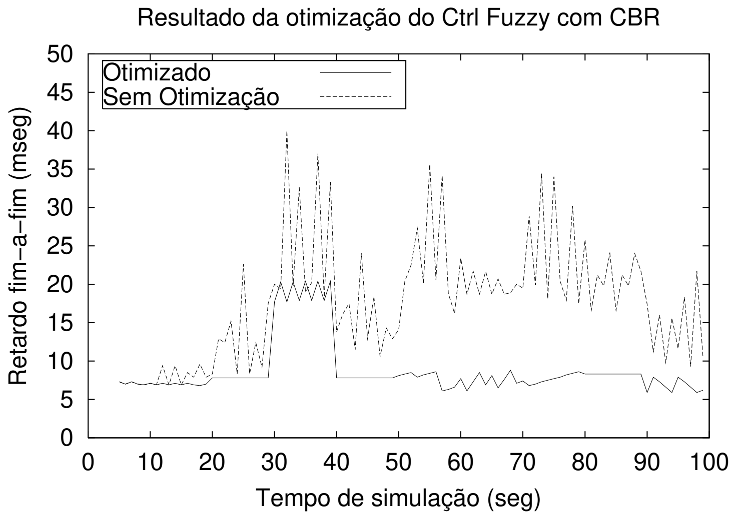
Precisamos, no entanto, definir as variáveis de otimização com critério, pois o otimizador por algoritmo genético não verifica o atendimento a uma determinada política e pode produzir uma função de pertinência que viole o contrato (SLA) do domínio. Para evitar esse problema, podemos estabelecer limites, definindo parâmetros não otimizáveis na descrição do controlador fuzzy.

5.4.2 Retardo obtido após otimização

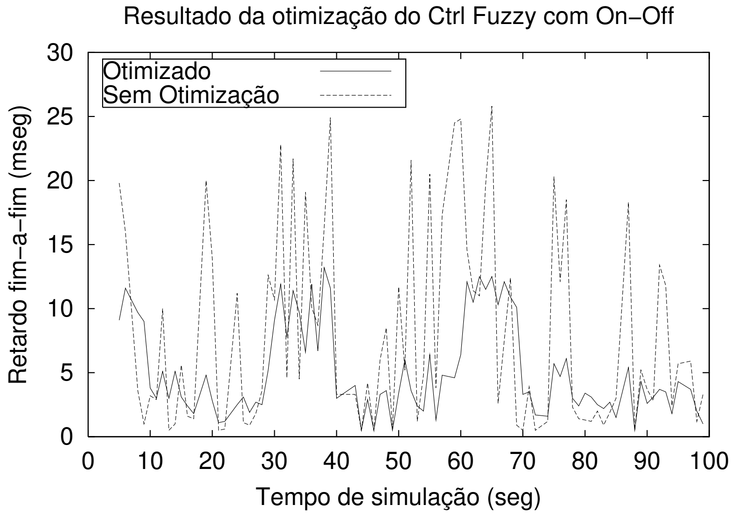
A eficácia do processo de otimização foi verificada comparando-se o retardo do tráfego EF, CBR e On-Off, na topologia simples, conforme figura 5.7. A curva original considera o controlador fuzzy antes do processo de otimização, mas com a base de regras verificada pelo algoritmo de Wang-Mendel. A curva otimizada considera o controlador após a otimização com uso do algoritmo genético.

Na figura 5.7(a), podemos observar a melhoria do retardo na classe EF após a realização do procedimento de otimização. Notamos, sobretudo, uma melhora sensível na segunda metade do período de simulação, onde os recursos de rede se tornam mais escassos. Na figura 5.7(b), podemos ver a melhoria do retardo na classe EF após a realização do procedimento de otimização, embora não tão acentuada quanto no caso anterior.

A melhoria do controlador para o tráfego CBR prejudica o desempenho para o tráfego On-Off. Como não podemos definir o modelo de tráfego *a priori*, optamos por escolher uma parametrização intermediária que mantivesse os valores de retardo próximos de 10 ms para ambos os padrões de tráfego. Vale lembrar que as taxas médias agregadas de tráfego CBR e On-Off são semelhantes em ambas as experiências.



(a) Tráfego CBR



(b) Tráfego On-Off

Figura 5.7: Retardo fim-a-fim de um fluxo EF com controlador fuzzy original e otimizado

5.5 Resultado da topologia simples

Apresentamos, nessa seção, os resultados da simulação em uma topologia simples. A topologia foi apresentada na seção 5.1.1 e o modelo de tráfego determinístico foi mostrado na seção 5.2.1. Para avaliar o desempenho do controlador fuzzy, comparamos com a situação de rede estaticamente provisionada e com um mecanismo de provisionamento dinâmico simples, apresentado na seção 4.6. O tempo exibido nos gráficos não considera os tempos de propagação das conexões, que por serem constantes, foram subtraídos para melhorar a visualização.

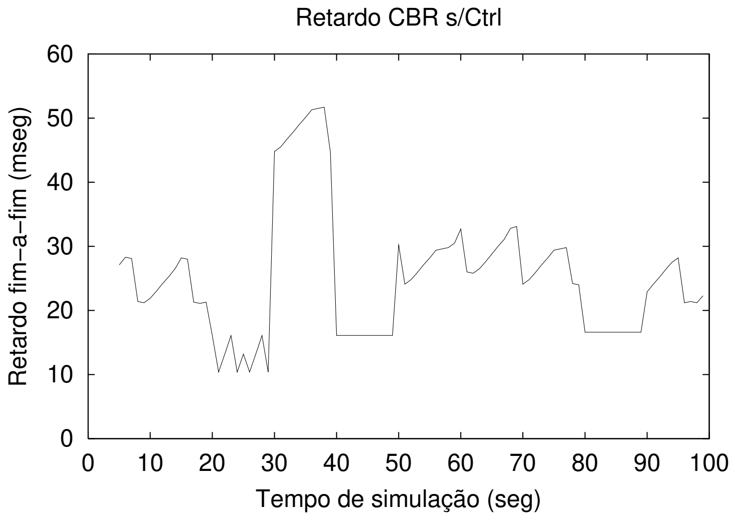
Todas as simulações iniciaram com uma configuração no escalonador de 50% da banda de saída, para cada uma das classes EF e BE. Para eliminar medidas de desempenho com a rede vazia, a seqüência de medidas iniciou-se sempre 5 segundos após o início do tráfego.

Os resultados apresentados aqui usaram o intervalo de 1 s entre as amostragens e, conseqüentemente, o intervalo de atuação do controlador. O tamanho do pacote escolhido foi de 576 bytes, tamanho que corresponde a 91,7% dos pacotes de um codificador G.711 a 64 Kbps [85]. Apresentamos, na seção 5.6, uma discussão sobre o comportamento dos diversos mecanismos de controle com a variação do intervalo de operação do controlador e do tamanho do pacote.

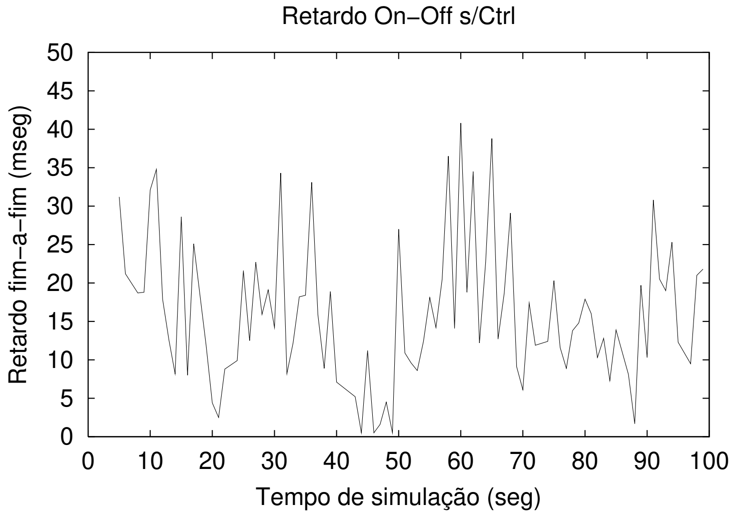
5.5.1 Retardo fim-a-fim da classe EF

A primeira medida avaliada é o retardo fim-a-fim de pacotes pertencentes à classe EF, desde a fonte até o destino do tráfego. Essa avaliação compara as situações sem controlador, com controlador convencional e com controlador fuzzy.

O gráfico da figura 5.8 exibe o resultado de um domínio DiffServ sem controlador. A figura 5.8(a) representa o tráfego CBR e a figura 5.8(b), o tráfego On-Off exponencial. Notamos, em 5.8(a), um pico de retardo, no intervalo 30 e 40 segundos porque nesse período a rede está no limite de sua capacidade, o retardo é alto mesmo com descarte alto de pacotes. No tráfego On-off, o retardo varia bastante ao longo da simulação, em função da distribuição exponencial das fontes de tráfego que entram no domínio.



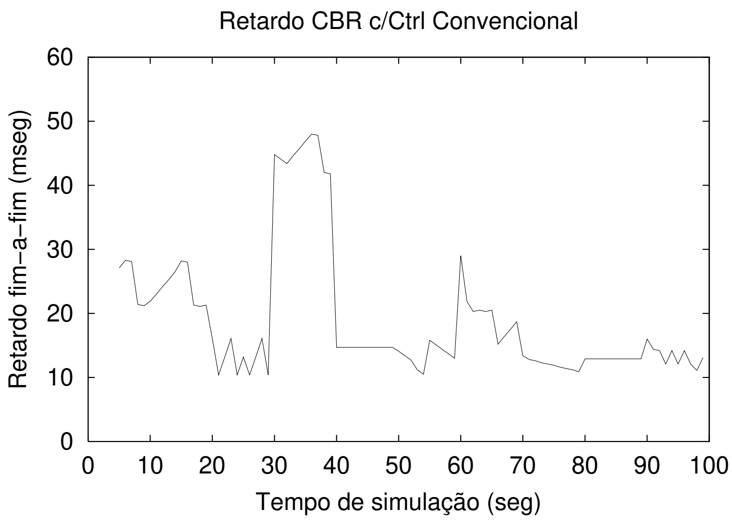
(a) Tráfego CBR



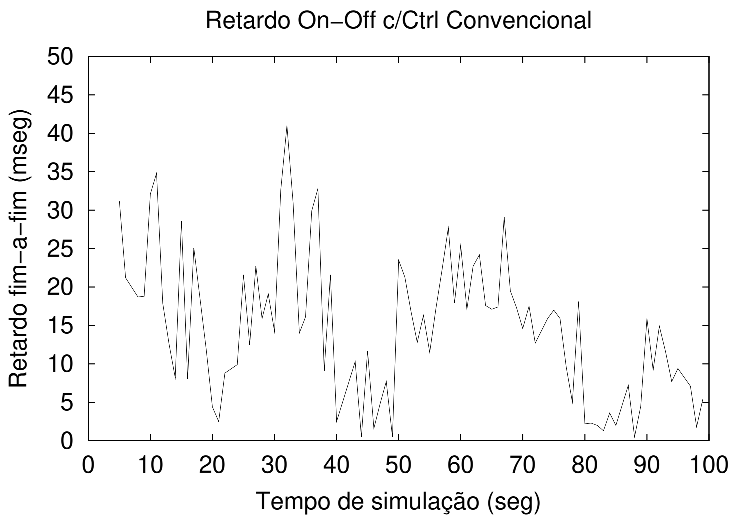
(b) Tráfego On-Off

Figura 5.8: Retardo fim-a-fim da classe EF sem controlador

O gráfico da figura 5.9 mostra o retardo fim-a-fim em um domínio DiffServ, com um controlador convencional. Apresentamos, na figura 5.9(a), o resultado com tráfego CBR e, na figura 5.9(b), o resultado do tráfego On-Off. Apesar de visualmente não notamos melhorias nas medidas de QoS, podemos verificar uma ligeira melhora nas medidas do controlador convencional, em relação à situação sem controlador, na tabela 5.2. Quando o sistema atinge o limite, o controlador corrige as variações, porém muito lentamente, não provocando melhoria sensível nesse cenário de tráfego agressivo.



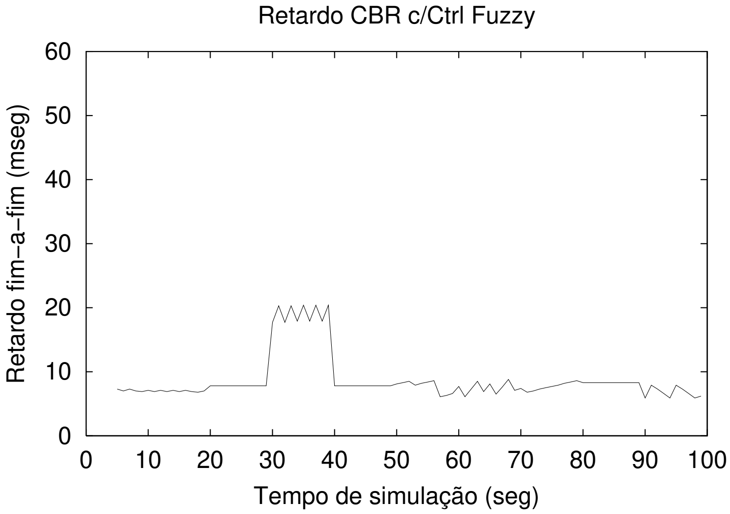
(a) Tráfego CBR



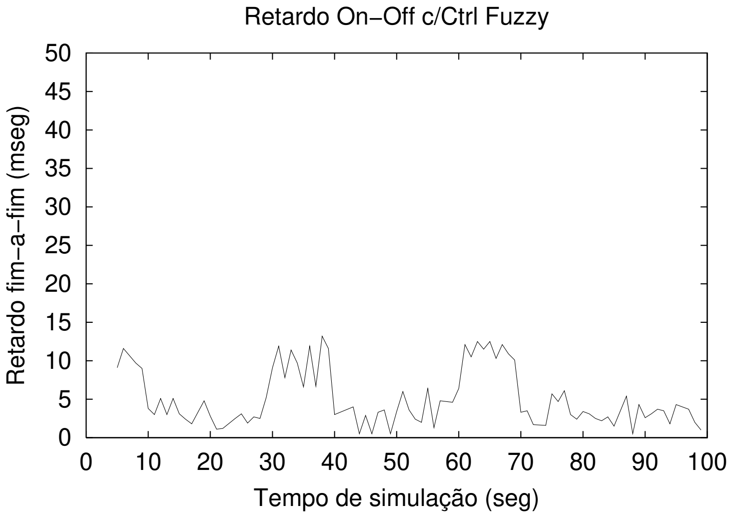
(b) Tráfego On-Off

Figura 5.9: Retardo fim-a-fim da classe EF com controlador convencional

O gráfico da figura 5.10 mostra o retardo fim-a-fim em um domínio DiffServ, com o controlador fuzzy. Notamos, em 5.10(a), melhoria do retardo em relação aos casos sem controlador e com controlador convencional. O controlador fuzzy corrige as variações rapidamente, mantendo os valores de retardo abaixo de 10 ms. Surge um aumento de retardo no períodos entre 30 s e 40 s porque, nesse momento, o tráfego EF ultrapassa o limite máximo de 90% da banda de saída, dedicada à sua classe. No tráfego On-off, figura 5.10(b), o retardo já é bem inferior aos verificados nos casos anteriores e os aumentos entre 30 s e 40 s e entre 60 s e 70 s também são devidos ao limite máximo da classe EF.



(a) Tráfego CBR



(b) Tráfego On-Off

Figura 5.10: Retardo fim-a-fim da classe EF com controlador Fuzzy

A tabela 5.2 apresenta os valores de medidas de retardo e variação de retardo médio e percentil 50, 90 e 95. Mostramos os resultados de simulação de tráfego CBR e On-Off sem o uso de controlador, usando o controlador convencional e usando o controlador fuzzy.

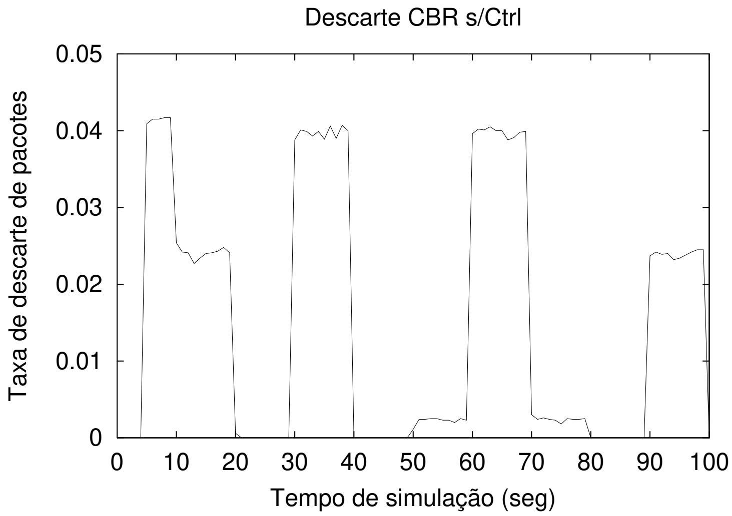
Tabela 5.2: Retardo e variação do retardo da classe EF na topologia simples (ms)

Tráfego	Média		Percentil 50		Percentil 90		Percentil 95	
	Retardo	Variação	Retardo	Variação	Retardo	Variação	Retardo	Variação
CBR S/Ctrl	43.4	1.7	43.0	0.4	65.8	0.4	75.3	7.3
CBR Conven.	33.6	1.5	30.3	0.4	58.8	0.4	71.3	6.0
CBR Fuzzy	19.3	1.0	17.3	0.4	34.9	0.4	41.1	0.4
OO S/Ctrl	17.9	1.2	16.6	0.3	33.2	3.6	36.9	5.7
OO Conven.	14.9	1.1	14.7	0.2	27.1	3.3	32.3	4.6
OO Fuzzy	5.6	0.5	3.7	0.1	11.7	1.3	12.4	1.9

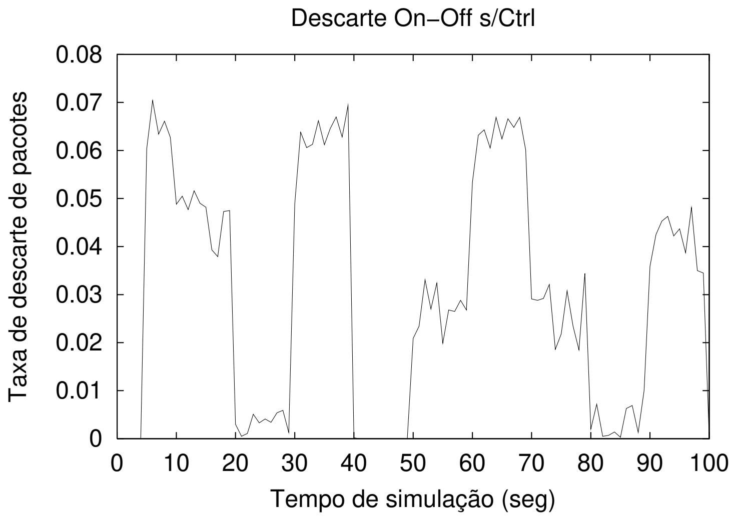
5.5.2 Descarte na classe EF

Apresentamos, a seguir, os gráficos da taxa de descarte de pacotes na classe EF, isto é, a relação pacotes descartados por pacotes transmitidos a cada segundo. O gráfico da figura 5.11 mostra a taxa de descarte do tráfego de voz da classe EF, ao longo da simulação. Sem controlador, o descarte com tráfego CBR ocorre em todo período em que se esgotam os recursos de rede, conforme mostrado na figura 5.11(a) e 5.11(b).

O gráfico da figura 5.12 mostra o descarte com o controlador convencional. Notamos uma diminuição do descarte da classe EF para ambos os tráfegos, em relação ao caso anterior.

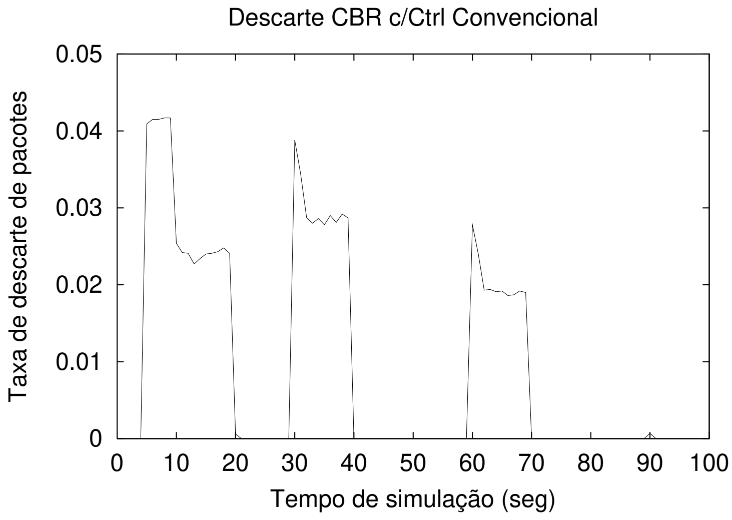


(a) Tráfego CBR

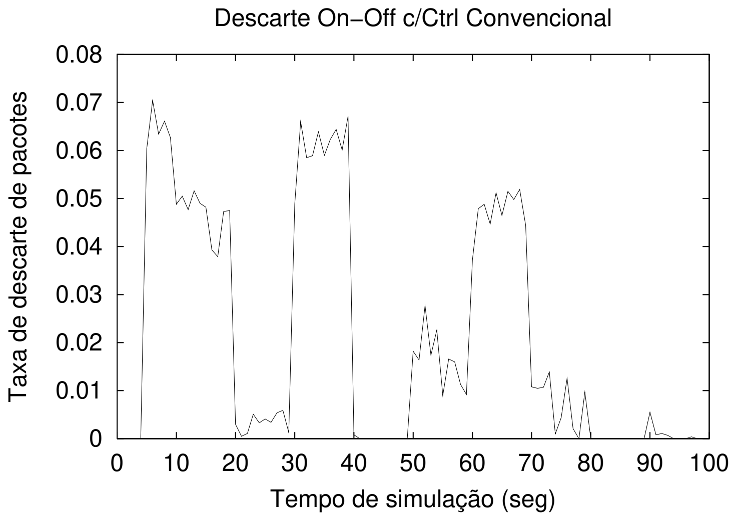


(b) Tráfego On-Off

Figura 5.11: Descarte na classe EF sem controlador



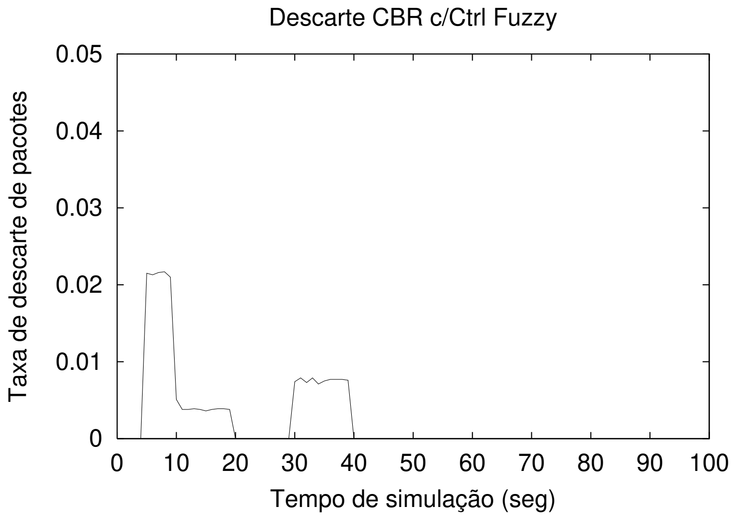
(a) Tráfego CBR



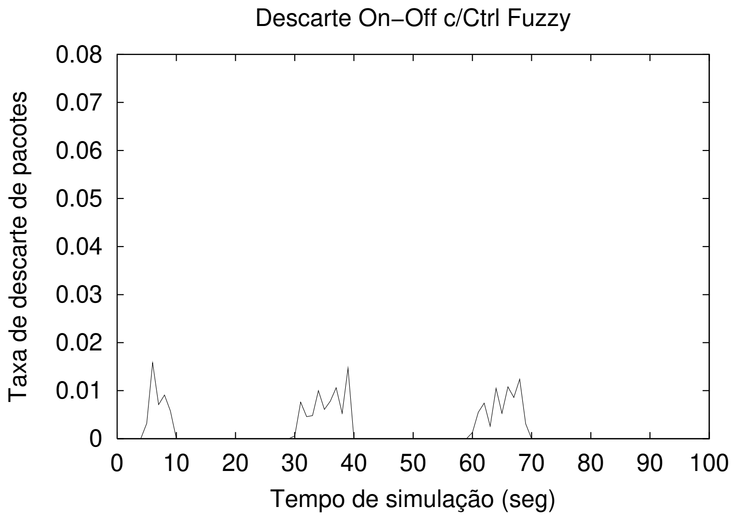
(b) Tráfego On-Off

Figura 5.12: Descarte na classe EF com controlador convencional

O gráfico da figura 5.13 mostra o descarte quando usamos o controlador fuzzy. Tanto no caso de tráfego CBR como tráfego On-Off, temos uma melhoria sensível na taxa de descarte.



(a) Tráfego CBR



(b) Tráfego On-Off

Figura 5.13: Descarte na classe EF com controlador Fuzzy

A tabela 5.3 apresenta o descarte de pacotes nas classes EF e BE, na topologia simples. Mostramos os resultados de simulação de tráfego CBR e On-Off sem o uso de controlador, usando o controlador convencional e usando o controlador fuzzy.

Podemos observar que há uma redução da taxa de descarte na classe EF com o controlador fuzzy comparado às situações sem controlador e com controlador convencional. Obviamente, quando se reduz a taxa da classe EF, provocamos um aumento

Tabela 5.3: Descarte na topologia simples

Controlador	EF (CBR)	BE (CBR)	EF(On-Off)	BE (On-Off)
Sem Ctrl	0.0198	0.0766	0.0379	0.1388
Convencional	0.0125	0.0843	0.0272	0.1439
Fuzzy	0.0068	0.0795	0.0027	0.1486

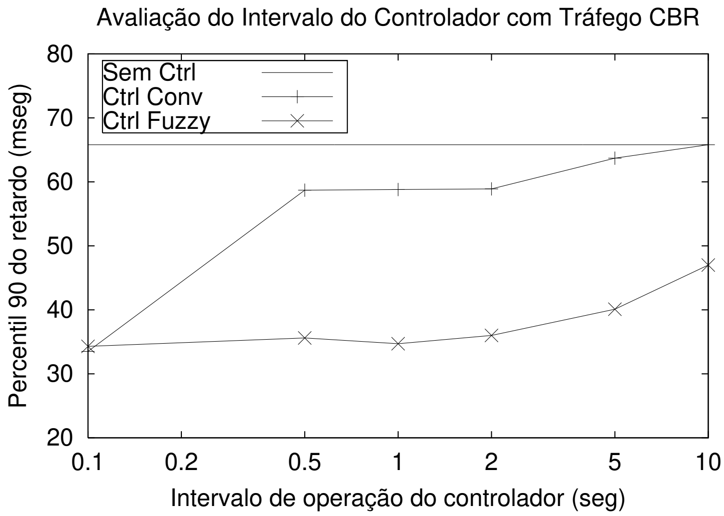
na taxa da classe BE, pois como o tráfego gerado é superior à capacidade da rede, haverá descartes. No entanto, podemos observar que o uso do controlador fuzzy diminui a taxa de descarte agregada, isto é, a soma das taxas das classes EF e BE é menor que as demais taxas agregadas, produzindo um melhor desempenho global.

5.6 Análise do resultado na topologia simples com variação de parâmetros

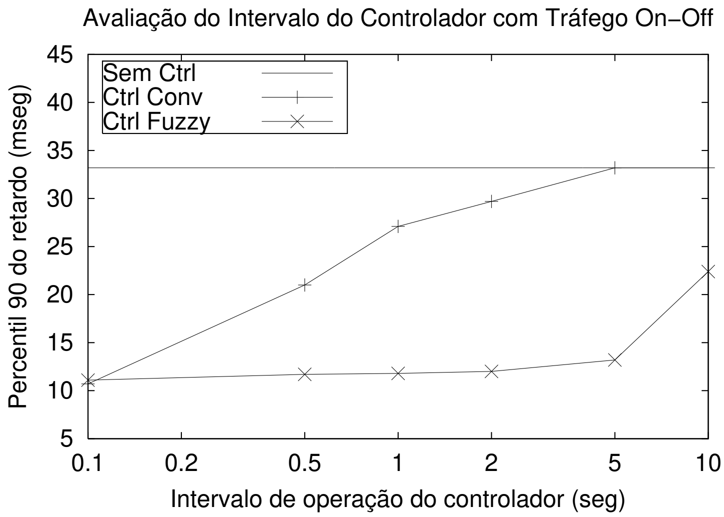
Foram realizadas medidas variando-se alguns parâmetros de rede para avaliar o comportamento do controlador fuzzy. Todas as medidas foram realizadas com a topologia simples e padrão de tráfego determinístico, conforme mostrado nas seções 5.1.1 e 5.2.1.

5.6.1 Avaliação da variação do intervalo de operação do controlador

O primeiro parâmetro avaliado foi o intervalo de operação do controlador que é mostrado na figura 5.14. A figura 5.14(a) mostra a avaliação do retardo na classe EF para tráfego CBR e a figura 5.14(b) mostra o retardo para tráfego On-Off. Medimos o resultado de percentil 90 do retardo variando o intervalo de operação do controlador em 0,1 s, 0,5 s, 1 s, 2 s, 5 s e 10 s. Utilizamos o tamanho de pacote 576 bytes. Para efeito de comparação, indicamos com uma linha contínua o percentil 90 do retardo da situação sem controlador.



(a) Tráfego CBR



(b) Tráfego On-Off

Figura 5.14: Avaliação do intervalo de atuação do controlador

As medidas do percentil 90 do retardo para o controlador convencional e fuzzy são próximas quando o intervalo é 0,1 s, em ambos os padrões de tráfego, porque a lentidão da resposta do controlador convencional não é observada com intervalo pequeno. Com intervalo maior que 10 s, no tráfego CBR, e maior que 2 s, no tráfego On-Off, o retardo do controlador convencional se aproxima do retardo do caso sem controlador, porque aqui a lentidão da resposta do controlador convencional não é desprezível, tornando o mesmo desnecessário. Esse comportamento é justificado porque a variação do tráfego ocorre a cada 10 segundos, conforme foi mostrado na figura 5.4.

Podemos observar que a resposta do controlador fuzzy é melhor para qualquer intervalo de operação, sendo praticamente constante para intervalos até 2 segundos e aproximadamente linear para intervalos a partir de 5 s. Esse comportamento é justificado pela variação do perfil do tráfego a cada 10 segundos (vide figura 5.4). Somente a partir desse intervalo (10 s), podemos notar uma degradação na resposta do controlador fuzzy.

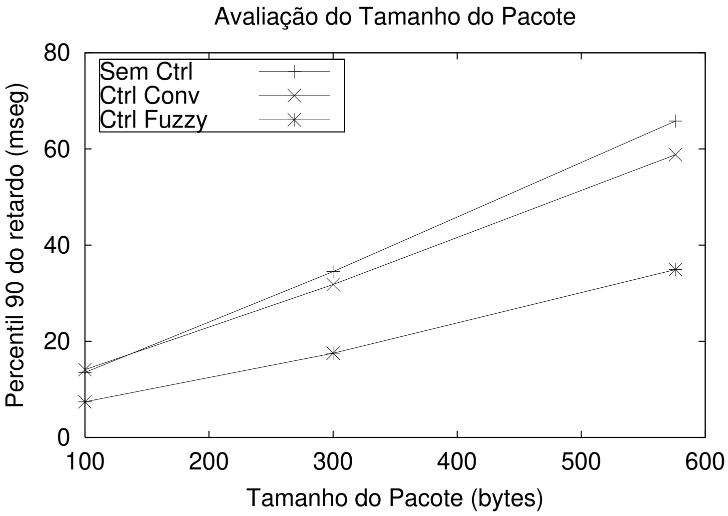
Podemos também notar que a diferença do retardo do tráfego On-Off obtido pelo controlador fuzzy, comparado com o caso sem controlador, é maior do que no caso com tráfego CBR, mostrando a maior eficiência do controlador fuzzy para tratar tráfego On-Off.

Assim, quando for utilizado um tráfego real, devemos recomendar um intervalo de operação do controlador proporcional ao período médio das conexões. Em nosso exemplo, como as conexões têm período de 10 segundos, o valor de até 5 segundos para operação do controlador fuzzy seria eficiente. O critério para a escolha do valor 1 segundos para intervalo de operação do controlador foi o equilíbrio entre resultado da QoS obtida e a sobrecarga computacional para ambos os controladores.

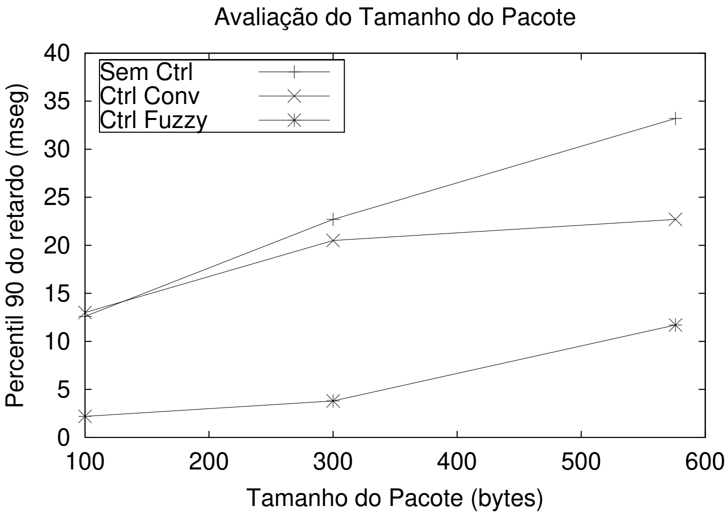
5.6.2 Avaliação da variação do tamanho do pacote IP

O segundo parâmetro avaliado foi o tamanho do pacote, mostrado na figura 5.15. A figura 5.15(a) mostra a avaliação do retardo na classe EF para tráfego CBR e a figura 5.15(b) mostra o retardo para tráfego On-Off. Variamos o tamanho do pacote desde 100 bytes, correspondente ao tráfego produzido por um codificador H.323, até 576 bytes, correspondente ao tráfego produzido por um codificador G.711. Os valores medidos foram o percentil 90 do retardo para pacotes com 100, 300 e 576 bytes. O tempo de operação do controlador foi de 1 segundo.

Na avaliação com tráfego CBR, mostrada na figura 5.15(a), podemos notar que com pacotes pequenos (100 bytes) os retardos estão próximos, apesar do controlador fuzzy já mostrar uma pequena superioridade. A justificativa para o retardo ser baixo com pacotes pequenos em qualquer controlador é o melhor intercalamento dos pacotes no escalonador. Com o aumento do tamanho dos pacotes o retardo aumenta quase linearmente. Para qualquer tamanho de pacote avaliado, o percentil 90 do retardo obtido com controlador fuzzy sempre foi inferior aos retardos obtidos pelo controlador convencional e no caso sem controlador, nesta ordem.



(a) Tráfego CBR



(b) Tráfego On-Off

Figura 5.15: Avaliação da influência do tamanho dos pacotes

Na avaliação com tráfego On-Off, mostrada na figura 5.15(b), podemos notar que o retardo obtido com o controlador fuzzy foi sempre inferior aos retardos obtidos com controlador convencional e sem controlador. Observamos também que o retardo, nesses dois últimos casos, é semelhante para pacotes pequenos, devido ao melhor intercalamento dos pacotes, conforme explicado anteriormente. Com pacotes maiores que 300 bytes, os retardos com controlador convencional são inferiores aos obtidos sem controlador.

Notamos também que a diferença do retardo com tráfego On-Off obtido pelo controlador fuzzy comparado com o controlador convencional e sem controlador, é maior que no caso com tráfego CBR, mais uma vez mostrando a melhor eficiência do controlador fuzzy para tratar tráfego On-Off.

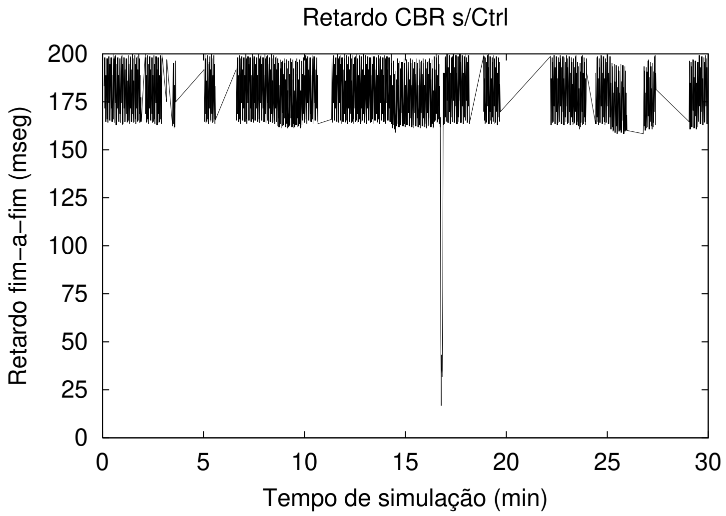
Finalmente, concluímos que o resultado obtido com o controlador fuzzy é melhor que o controlador convencional e sem controlador para qualquer tamanho de pacote. O critério para escolha do tamanho 576 bytes foi o tamanho típico dos pacotes produzidos por um codificador G.711.

5.7 Resultado da topologia complexa

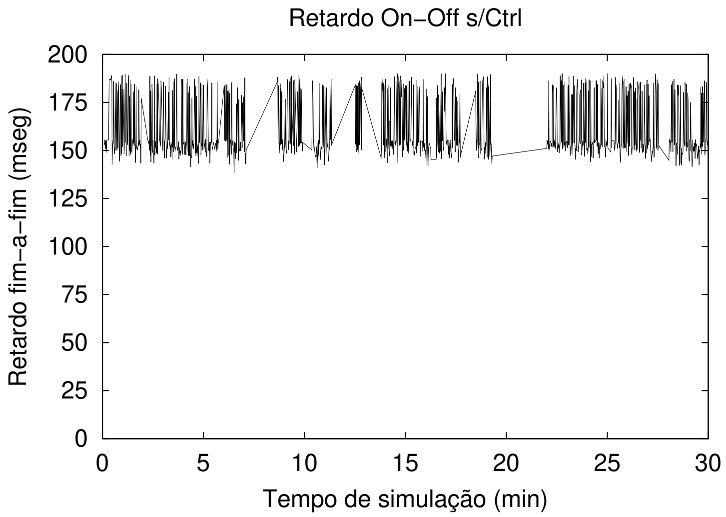
Apresentamos, nessa seção, as simulações com topologia complexa, mostrada na seção 5.1.2, com o modelo de tráfego complexo, mostrado na seção 5.2.2. As medições realizadas foram retardo e percentil do retardo fim-a-fim e variação do retardo (*jitter*) da classe EF, durante um período de 30 minutos. São mostrados os gráficos de um fluxo qualquer e a tabela de percentil de retardo e variação do retardo sem a utilização de controlador, com controlador convencional e com o controlador fuzzy. Todas as simulações iniciam com alocação de 50% da banda de saída para cada classe e, para eliminar medidas com a rede sem tráfego, somente após 30 segundos começa as seqüência de medidas.

5.7.1 Retardo fim-a-fim da classe EF

O gráfico da figura 5.16 mostra o retardo fim-a-fim de um tráfego de voz da classe EF, sem controlador. Esse gráfico foi extraído de um exemplo com a topologia 1. O gráfico mostra apenas o tempo das filas pois o tempo de transmissão dos links, por ser constante, foi subtraído. A figura 5.16(a) foi obtida com tráfego CBR e a figura 5.16(b), com tráfego On-Off exponencial.



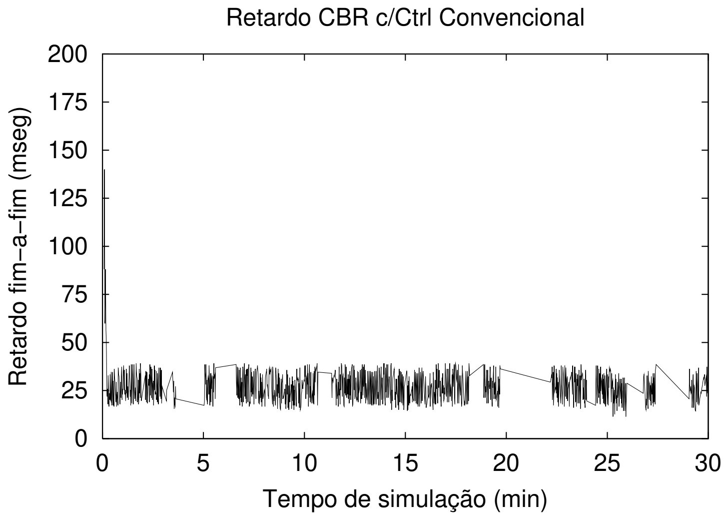
(a) Tráfego CBR



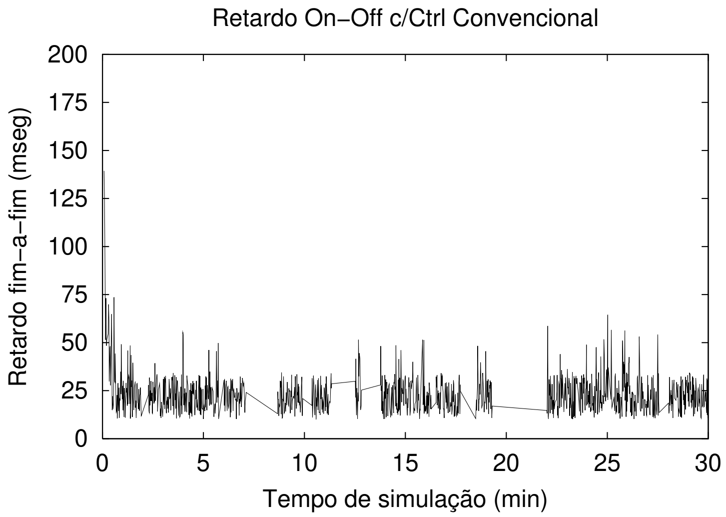
(b) Tráfego On-Off

Figura 5.16: Retardo fim-a-fim de um fluxo da classe EF, sem controlador, na topologia 1

O gráfico da figura 5.17 mostra o retardo fim-a-fim com um controlador convencional. Observamos na figura 5.17(a), melhoria do retardo comparado com o caso sem controlador. O tráfego On-Off também apresenta uma melhora.



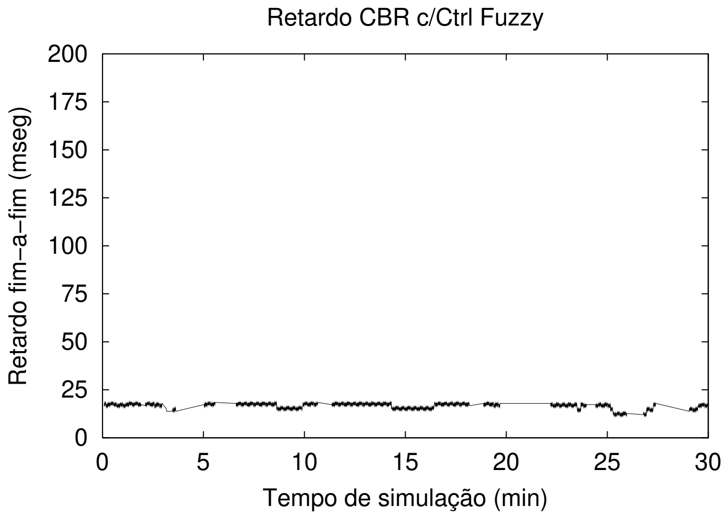
(a) Tráfego CBR



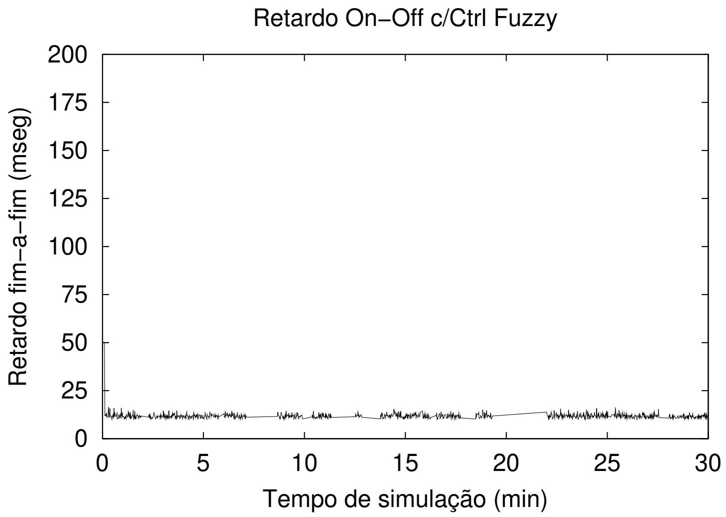
(b) Tráfego On-Off

Figura 5.17: Retardo fim-a-fim de um fluxo da classe EF, com controlador convencional, na topologia 1

O gráfico da figura 5.18 mostra o retardo fim-a-fim com um controlador convencional. Observamos, na figura 5.18(a), melhoria do retardo comparado com o caso sem controlador e com controlador convencional. Para o tráfego On-Off, houve também uma melhora. O controlador fuzzy corrige as variações do tráfego mantendo o retardo baixo. No caso de tráfego On-Off, na figura 5.18(b),o retardo médio é menor que nos casos anteriores.



(a) Tráfego CBR



(b) Tráfego On-Off

Figura 5.18: Retardo fim-a-fim de um fluxo da classe EF, com controlador Fuzzy, na topologia 1

A tabela 5.4 mostra o retardo e a variação de retardo do tráfego CBR e On-Off para a topologia 1. As tabelas 5.5 e 5.4 mostram o retardo e variação de retardo para tráfego CBR e On-Off, para as topologias 2 e 3, respectivamente.

Podemos observar que o controlador fuzzy se mostrou eficiente para melhorar a QoS em diferentes domínios DiffServ, com características da Internet, oferecendo

Tabela 5.4: Retardo e variação do retardo da classe EF na topologia 1 (ms)

Tráfego	Média		Percentil 50		Percentil 90		Percentil 95	
	Retardo	Variação	Retardo	Variação	Retardo	Variação	Retardo	Variação
CBR S/Ctrl	200.6	20.3	203.3	0.0	240.7	71.9	252.2	72.1
CBR Conven.	45.8	1.9	44.9	0.0	66.8	0.0	72.0	0.0
CBR Fuzzy	35.1	1.1	33.9	0.0	54.7	0.0	58.7	0.0
OO S/Ctrl	161.1	15.4	153.7	8.7	184.2	35.0	186.5	37.8
OO Conven.	25.3	12.2	23.4	9.1	40.3	27.1	50.1	35.6
OO Fuzzy	11.0	3.6	11.0	3.1	14.3	6.6	15.0	7.7

Tabela 5.5: Retardo e variação do retardo da classe EF na topologia 2 (ms)

Tráfego	Média		Percentil 50		Percentil 90		Percentil 95	
	Retardo	Variação	Retardo	Variação	Retardo	Variação	Retardo	Variação
CBR S/Ctrl	198.7	19.8	201.5	0.0	239.0	71.9	249.9	72.1
CBR Conven.	43.9	1.7	43.1	0.0	65.0	0.0	70.2	0.0
CBR Fuzzy	33.2	1.0	32.2	0.0	52.4	0.0	56.4	0.0
OO S/Ctrl	159.3	13.8	151.4	6.8	182.3	34.2	184.2	36.4
OO Conven.	23.4	12.0	21.6	9.0	38.5	26.9	47.7	35.5
OO Fuzzy	9.2	2.8	8.9	2.2	11.8	5.0	12.5	5.9

retardo menor que nas situações de provisionamento estático (sem controlador) e com controlador convencional. A proximidade dos valores de retardo do controlador convencional e fuzzy pode ser justificada pelo fato do tempo de conexão médio do perfil de tráfego ser da ordem de 1 a 3 minutos, muito maior que o intervalo de operação do controlador (1 segundo). Esse comportamento foi mostrado na avaliação do intervalo de operação, apresentado na seção 5.6.1. Porém, devemos observar que a variação de retardo (*jitter*)do controlador fuzzy é menor, propiciando uma melhor QoS para tráfego de telefonia IP.

Tabela 5.6: Retardo e variação do retardo da classe EF na topologia 3 (ms)

Tráfego	Média		Percentil 50		Percentil 90		Percentil 95	
	Retardo	Variação	Retardo	Variação	Retardo	Variação	Retardo	Variação
CBR S/Ctrl	350.3	35.2	405.5	0.0	458.4	143.9	470.5	213.6
CBR Conven.	131.9	67.2	150.9	69.6	194.1	143.9	202.7	143.9
CBR Fuzzy	114.6	57.6	130.1	69.6	169.3	143.9	175.1	143.9
OO S/Ctrl	262.9	28.9	295.3	4.4	334.9	51.0	338.7	174.4
OO Conven.	58.9	28.1	62.3	22.4	87.3	63.0	93.3	70.5
OO Fuzzy	38.6	22.0	48.8	23.0	53.5	43.2	54.7	44.8

5.7.2 Descarte da classe EF

A tabela 5.7 mostra a taxa de descarte de pacotes pertencentes ao tráfego de voz, na classe EF, e o tráfego concorrente da classe BE na topologia 1. A tabela 5.8 mostra a taxa de descarte, na topologia 2, e a tabela 5.9, na topologia 3. Podemos notar uma redução na taxa de descarte com controlador fuzzy, para ambos os tipos de tráfego, comparando com os casos sem controlador e com controlador convencional.

Como o tráfego gerado é superior à capacidade da rede, a redução da taxa de descarte na classe EF provoca um aumento na taxa da classe BE. No entanto, podemos observar que o uso do controlador fuzzy diminui a taxa de descarte agregada, isto é, a soma das taxas das classes EF e BE é menor que nos outros casos, obtendo-se portanto melhor desempenho global.

Tabela 5.7: Descarte na topologia 1 (Pacotes descartados/transmitidos)

Controlador	EF (CBR)	BE (CBR)	EF(On-Off)	BE (On-Off)
Sem Ctrl	0.0195	0.0275	0.0293	0.0349
Convencional	0.0002	0.0423	0.0002	0.0573
Fuzzy	0.0000	0.0422	0.0000	0.0557

Tabela 5.8: Descarte na topologia 2 (Pacotes descartados/transmitidos)

Controlador	EF (CBR)	BE (CBR)	EF(On-Off)	BE (On-Off)
Sem Ctrl	0.0206	0.0302	0.0320	0.0378
Convencional	0.0003	0.0454	0.0003	0.0624
Fuzzy	0.0000	0.0454	0.0000	0.0608

Tabela 5.9: Descarte na topologia 3 (Pacotes descartados/transmitidos)

Controlador	EF (CBR)	BE (CBR)	EF(On-Off)	BE (On-Off)
Sem Ctrl	0.0336	0.0458	0.0423	0.0468
Convencional	0.0084	0.0653	0.0104	0.0715
Fuzzy	0.0066	0.0667	0.0052	0.0731

5.8 Comentários

Apresentamos, neste capítulo, as topologias de simulação e os modelos de tráfego utilizados e as medidas avaliadas para verificar o funcionamento do controlador proposto. Mostramos, também, os resultados da simulação em uma topologia simples e em uma topologia complexa. Em cada seção, foram mostrados os resultados de retardo fim-a-fim e descarte de pacotes, além dos valores de percentil de retardo fim-a-fim e taxa de descarte.

Apresentamos também uma avaliação do comportamento dos controladores utilizados quando se variou o intervalo de operação do controlador e o tamanho dos pacotes transmitidos. A variação de intervalo de operação do controlador mostrou que, o controlador fuzzy propicia um melhor resultado comparado com o controlador convencional, isto é, podemos aumentar o intervalo de operação do controlador fuzzy sem degradar a QoS. A variação do tamanho do pacote mostrou que o controlador fuzzy apresenta melhor QoS com todos os tamnhos avaliados.

Em todas as situações, foi demonstrada a eficiência do controlador fuzzy proposto, através da verificação da funcionalidade do protótipo, realizada em simulação. Como resultado final, temos a validação da metodologia apresentada nos capítulos anteriores.

Capítulo 6

Conclusões e trabalhos futuros

APRESENTAMOS , nesse trabalho, uma proposta de metodologia de construção de um mecanismo de provisionamento dinâmico, para a melhoria da qualidade de serviço em uma arquitetura DiffServ. Essa metodologia propõe um controlador que reconfigura dinamicamente os parâmetros dos equipamentos de redes, melhorando a QoS no âmbito de um domínio. A utilização de lógica fuzzy no controlador possibilitou tratamento eficaz das imprecisões e incertezas do tráfego de entrada no domínio. O uso dos algoritmos Wang-Mendel e genético propiciaram uma otimização nos parâmetros do controlador e a obtenção de um resultado ótimo, sem depender dos critérios de escolha do projetista. O controlador, ainda assim, mantém sua complexidade baixa, não comprometendo a característica de escalabilidade da arquitetura DiffServ.

O algoritmo genético exige muito recurso computacional, mas ele não influencia os equipamentos de rede, pois a otimização é realizada no equipamento gerenciador. Além disso, seu funcionamento é esporádico (da ordem de horas ou dias), o que reduz a necessidade computacional. Em nossos experimentos, a otimização de um controlador previamente otimizado, com pequenas variações no ambiente, consumiu poucos minutos de processamento.

O gerenciamento baseado em políticas permite que o controlador seja genérico, podendo definir o comportamento do sistema de acordo com as decisões administrativas do gerenciador do domínio. Além disso, o comportamento pode ser facilmente alterado em todo o domínio, durante o funcionamento normal do sistema. A proposta de mapeamento de políticas *Ponder* em parâmetros de controlador fuzzy demonstra a capacidade da lógica fuzzy de representar requisitos de QoS abstratos.

As simulações realizadas para validar o modelo utilizaram exemplo com classes EF e BE. Apesar de conceitualmente simples, as avaliações de retardo e variação de retardo são dificultadas pela imprecisão e incerteza do tráfego que entra no domínio[50]. Os resultados obtidos através de simulação demonstram a viabilidade da proposta, pela sensível melhoria nas medidas da qualidade de serviço, comparadas àquelas das situações sem controlador ou com controlador convencional. Os resultados mostram também melhoria de QoS no caso de um domínio maior, com nós criados aleatoriamente. Esse exemplo demonstra a funcionalidade do modelo em uma situação semelhante à real, com topologias e geração de tráfego aleatórios. A experiência com topologias aleatórias mostrou que o modelo pode ser generalizado para qualquer topologia real.

Foram realizadas simulações variando-se o intervalo de operação do controlador e o tamanho dos pacotes transmitidos. Quando variamos o intervalo de operação do controlador mostramos que, com intervalo pequeno, o resultado do controlador fuzzy é apenas um pouco melhor que o do controlador convencional, tornando-se mais significativo em intervalos maiores. O controlador fuzzy mostrou-se mais eficiente que o convencional para qualquer intervalo de operação. Outra observação é que o intervalo ideal é proporcional ao tempo médio das conexões dos fluxos que cruzam o domínio. Comparando os resultados de tráfego CBR e On-Off notamos que o controlador fuzzy apresenta-se mais eficiente comtráfego On-Off, ou seja, tráfegos encontrados em uma rede real.

Quando variamos o tamanho do pacote, observamos que, com pacotes pequenos, as medidas de QoS são semelhantes para todas as situações. Esse fato, já conhecido nas redes ATM, demonstra que reduzir os tamanhos dos pacotes é uma medida eficaz para melhorar a QoS. Com o aumento do tamanho do pacote, observamos uma degradação na QoS em todas as situações, porém o controlador fuzzy continua apresentando o melhor resultado. Novamente podemos observar que o controlador fuzzy produz um melhor resultado comparativo quando trata tráfego On-Off em relação ao tráfego CBR, confirmando sua melhor eficiência para tratar tráfegos reais.

Finalmente, podemos concluir que mecanismos de provisionamento dinâmico apresentam vantagens em manter QoS, quando ocorre variações normais nos fluxos de tráfego. A atitude usual de superprovisionamento possibilita a manutenção da qualidade, porém o custo para manter essa infra-estrutura extra é alto. Mostramos também que a lógica fuzzy se mostrou adequada para tratar tráfegos com alta variação, mais comuns nas situações reais. A arquitetura proposta permitiu a construção de um sistema escalável e eficiente. A utilização de gerenciamento baseado em políticas possibilitou a facilidade da especificação dos requisitos de QoS desejados.

Como continuação do trabalho, poderá ser definido um controlador que inclua suporte a outras classes DiffServ, como AF *(Assured Forwarding)*. Essa classe segue filosofia diferente, devendo o controlador tratar variáveis distintas das consideradas neste trabalho. Assim, teremos um controlador completo, com capacidade de ajustar dinamicamente os parâmetros de todas as classes DiffServ, de acordo com as variações do tráfego e políticas especificadas.

Outro desafio é o tratamento de redes móveis. Como é previsto que qualquer infra-estrutura de redes em um futuro próximo tenha segmentos móveis e aéreos, seria interessante avaliar a aplicação da metodologia aqui proposta nesse caso. O tratamento de QoS em ambiente móvel ainda é um problema em aberto.

Outro trabalho futuro deve ser a avaliação de desempenho, tanto nos equipamentos de rede como no gerenciador central. Como todas as avaliações foram realizadas em simulação, não foi possível avaliar o impacto dos novos mecanismos adicionados à rede. No caso dos equipamentos de rede, a única função nova é o controlador fuzzy, considerando que o equipamento já disponha de protocolo de gerenciamento SNMP/COPS. Como esse controle é realizado com frequência alta (da ordem de segundos), a sobrecarga adicionada pode ser prejudicial ao desempenho do equipamento.

Uma proposta para melhorar o desempenho de controlador fuzzy é implementá-lo em hardware (silício), no lugar da forma tradicional em software. Essa proposta, feita por Catania *et al.*[56, 57] demonstra que a implementação de controlador fuzzy em silício torna praticamente imperceptível o impacto no desempenho do equipamento. No entanto, essa metodologia não pode ser implementada diretamente em nossa aplicação, porque o controlador fuzzy deve ser reconfigurado durante a operação da rede. Assim, a utilização dessa metodologia em nossa proposta exigiria a utilização de algum mecanismo de "reconfiguração de hardware".

Referências Bibliográficas

[1] Blake, S., Black, D., and Carlson, M. "An architecture for differentiated services". RFC 2475, dezembro de 1998.

[2] Blight, D., and Hamada, T. "Policy-based networking architecture for qos interworking in ip management". In: *Integrated Network Management, Distributed Management for the Networked Millennium 1999*, 1999, pp. 811–826.

[3] Lee, C. C. "Fuzzy Logic in control systems: Fuzzy logic controller, Part II", *IEEE Transactions on Systems, Man, and Cybernetics v. 20*, n. 2, março de 1990, pp. 419–435.

[4] Vasilakos, A., and Anagnostakis, K. "Evolutionary-fuzzy prediction for strategic inter-domain routing: architecture and mechanisms". In: *WCCI 98*, Anchorage, EUA, maio de 1998.

[5] Herrera, F., Lozano, M., and Verdegay, J. "Tuning fuzzy logic controllers by genetic algorithms", *International Journal of Approximate Reasoning v. 12*, n. 3, junho de 1995, pp. 299–315.

[6] Kim, J., Moon, Y., and Zeigler, B. P. "Designing fuzzy net controllers using GA optimization". In: *Proceedings IEEE/IFAC Joint Symposium on Computer-Aided Control System Design*, Tucson, USA, março de 1994, pp. 83–88.

[7] Velasco, J., and Magdalena, L. "Genetic algorithms in fuzzy control systems". In: *Genetic Algorithms in Engineering and Computer Science*, G. Winter, J. Periaux, M. Galan, and P. Cuesta, Eds. John Wiley & Sons, 1995, pp. 141–165.

[8] Trimintzios, P., Pavlou, G., Andrikopoulos, I., Griffin, D., Jacquenet, C., Georgatsos, P., T'joens, Y., Georgiadis, L., Egan, R., and

Memenios, G. "An architectural framework for providing qos in ip differential service networks". In: *VII IFIP/IEEE International Symposium on Integrated Network Management (IM 2001)*, 2001.

[9] Pavlou, G., Andrikopoulos, I., Griffin, D., Jacquenet, C., Georgatsos, P., T'joens, Y., Georgiadis, L., Egan, R., and Memenios, G. "Service level specification semantics, parameters and negotiation requirements". Internet Draft draft-tequila-diffserv-sls-00.txt, julho de 2001.

[10] Flegkas, P., Trimintzios, P., Pavlou, G., Andrikopoulos, I., Griffin, D., Jacquenet, C., Georgatsos, P., T'joens, Y., Georgiadis, L., Egan, R., and Memenios, G. "On policy-based extensible hierarchical network management in QoS-enable IP networks". In: *Workshop on Policies for Distributed Systems and Network (Policy 2001)*, Bristol, UK, junho de 2001.

[11] Westberg, L., Jacobsson, M., Karagiannis, G., and Oosthoek, S. "Resource management in diffserv (RMD) framework". Internet Draft draft-westberg-rmd-framework-00.txt, abril de 2001.

[12] Westberg, L., Jacobsson, M., Karagiannis, G., and Oosthoek, S. "Resource management in Diffserv on demand (RODA) PHR". Internet Draft draft-westberg-rmd-od-phr-00.txt, abril de 2001.

[13] Liao, R. R., and Campbell, A. T. "Dynamic edge provisioning for core networks". In: *IFIP/IEEE Eighth International Workshop on Quality of Service (IWQoS 2000)*, Pittsburgh, USA, junho de 2000.

[14] Liao, R. R., and Campbell, A. T. "Dynamic core provisioning for quantitative differentiated service". In: *IFIP/IEEE Ninth International Workshop on Quality of Service (IWQoS 2001)*, Karlsruhe, Germany, junho de 2001.

[15] Fernandez, M. P., de Castro P. Pedroza, A., and de Rezende, J. F. "Qualidade de serviço em um domínio diffserv através de gerenciamento baseado em políticas". In: *XIX Simpósio Brasileiro de Redes de Computadores (SBRC'2001)*, Florianópolis, Brasil, maio de 2001.

[16] Fernandez, M. P., de Castro P. Pedroza, A., and de Rezende, J. F. "Quality of service in a diffserv domain using policy-based management". In: *Teletraffic Engeneering in the Internet Era*, 1 ed., v. 4. Elsevier Science B.V., Amsterdam, The Netherlands, setembro de 2001, ch. QoS IP - Diff Service, pp. 919–930. ISBN 0-444-50911-9.

[17] Fernandez, M. P., de Castro P. Pedroza, A., and de Rezende, J. F. "Policy-based management providing qos in a diffserv domain". In: *Latin American Network Operation and Management Symposium (LANOMS'2001)*, Belo Horizonte, Brazil, agosto de 2001.

[18] Fernandez, M. P., de Castro P. Pedroza, A., and de Rezende, J. F. "Qos provisioning across a diffserv domain using policy-based management". In: *(Globecom 2001)*, San Antonio, USA, novembro de 2001.

[19] Fernandez, M. P., de Castro P. Pedroza, A., and de Rezende, J. F. "Otimização de controlador fuzzy para provisionamento de recursos em ambiente diffserv através de algoritmo genético". In: *XX Simpósio Brasileiro de Redes de Computadores (SBRC'2002)*, Búzios, Brasil, maio de 2002.

[20] Fernandez, M. P., de Castro P. Pedroza, A., and de Rezende, J. F. "Optimizing fuzzy controllers with genetic algorithms for qos improvement". In: *International Telecommunication Symposium (ITS'2002)*, Natal, Brazil, setembro de 2002.

[21] Fernandez, M. P., de Castro P. Pedroza, A., and de Rezende, J. F. "Dynamic qos provisioning in diffserv domains using fuzzy logic controllers", *Telecommunication Systems Journal*, julho de 2002. Submetido para publicação.

[22] Braden, R., zhang, L., Berson, S., Herzog, S., and Jamin, S. "Resorce ReSerVation Protocol (RSVP) - version 1 Funtional Specification". RFC 2205, setembro de 1997.

[23] Nichols, K., and Carpenter, B. "Definition of differentiated services per domain behaviors and rules for their specification". RFC 3086, abril de 2001.

[24] Black, D., Brim, S., Carpenter, B., and Faucheur, F. L. "Per hop behavior identification codes". RFC 3140, junho de 2001.

[25] Jacobson, V., Nichols, K., and Poduri, K. "An expedited forwarding PHB". RFC 2598, junho de 1999.

[26] Heinanen, J., Baker, F., Weiss, W., and WroclawskiV, J. "Assured forwarding PHB group". RFC 2597, junho de 1999.

[27] Floyd, S., and Jacobson, V. "Random early detection gateways for congestion avoidance", *IEEE/ACM Transactions on Networking v. 1*, n. 4, agosto de 1993, pp. 397–413.

[28] Huston, G. "Next Steps for the IP QoS Architecture". RFC 2990, novembro de 2000.

[29] Trimintzios, P., Andrikopoulos, I., Pavlou, G., Flegkas, P., Griffin, D., Georgatsos, P., Goderis, D., T'Joens, Y., Georgiadis, L., Jacquenet, C., and Egan, R. "A management and control architecture for providing IP differentiated services in MPLS-based networks", *IEEE Communications Magazine v. 39*, n. 5, maio de 2001, pp. 80–88.

[30] Westberg, L., Turanyi, Z., and Partain, D. "Load control of real-time traffic". Internet Draft draft-westberg-rmd-framework-00.txt, outubro de 2000.

[31] Jacobson, M., Karagiannis, G., de KoGel, M., Oosthoek, S., Partain, D., Rexhepi, V., and Wallentin, P. "Resource Management in Diffserv On DemAnd (RODA) PHR". Internet Draft, 2002.

[32] Semret, N., Liao, R. R., Campbell, A. T., and Lazar, A. "Peering and provisioning of differentiated Internet services". In: *IEEE INFOCOM 2000*, Tel-Aviv, Israel, março de 2000.

[33] Semret, N., Liao, R. R., Campbell, A. T., and Lazar, A. "Pricing, provisioning and peering: Dynamic markets for differentiated internet services and implications for network interconnections", *IEEE Journal on Selected Areas in Communications v. 18*, n. 12, dezembro de 2000, pp. 2499–2513. Special Issue on Internet QoS.

[34] Sloman, M. "Policy driven management for distributed systems", *Journal of Networking and Systems Management v. 2*, n. 4, 1994, pp. 333–360. Plenum Press.

[35] Moffett, J., and Sloman, M. "Policy hierarchies for distributed system management", *IEEE Journal on Selected Areas in Communications v. 11*, n. 9, 1993, pp. 1404–1414. IEEE.

[36] Lupu, E., and Sloman, M. "Towards a role based framework for distributed systems management", *Journal of Network and Systems Management v. 5*, n. 1, janeiro de 1997, pp. 5–30. Plenum Press Publishing.

[37] Lupu, E., and Sloman, M. "Conflicts in policy-based distributed systems management", *IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management v. 25*, n. 6, 1999, pp. 852–869. IEEE.

[38] Fine, M., McCloghrie, K., Seligson, J., Chan, K., Hahn, S., Smith, A., and Reichmeyer, F. "Differentiated services quality of service policy information base". Internet Draft draft-ietf-diffserv-pib-02.txt, novembro de 2000.

[39] Snir, Y., Ramberg, Y., Strassner, J., and Cohen, R. "Policy QoS information model". Internet Draft draft-ietf-policy-qos-info-model-03.txt, abril de 2001.

[40] Moore, B., Durham, D., Halpern, J., Strassner, J., Westerinen, A., and Weiss, W. "Information model for describing network device qos datapath mechanisms". Internet Draft draft-ietf-policy-qos-device-info-model-05.txt, julho de 2001.

[41] Rajan, R., Verma, D., and Kamat, S. "A policy framework for integrated and differentiated services in the Internet", *IEEE Network*, 1999, pp. 36–41. Plenum Press.

[42] Moore, B., Ellesson, E., Strassner, J., and Westerinen, A. "Policy core information model – version 1 specification". RFC 3060, fevereiro de 2001.

[43] Stevens, M., Weiss, W., and Mamon, M. "Policy framework". Internet Draft draft-ietf-policy-framework-00.txt, setembro de 1999.

[44] Mamon, M., Bernet, Y., and Herzog, S. "Requirements for a policy management system". Internet Draft draft-ietf-policy-req-02.txt, novembro de 2000.

[45] Moore, B., Strassner, J., and Elleson, E. "Policy core information model". Internet Draft draft-ietf-policy-core-info-model-08.txt, outubro de 2000.

[46] Durham, D., Boyle, J., and Coen, R. "The COPS (Common Open Policy Service) protocol". RFC 2748, janeiro de 2000.

[47] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. "Ponder Policy Specification Language". http://www-dse.doc.ic.ac.uk/policies/ponder.shtml.

[48] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. "The Ponder policy specification language". In: *Policy 2001: Workshop on Policies for Distributed Systems ans Networks*, Bristol, UK, janeiro de 2001, pp. 18–39.

[49] Guérin, R., and Orda, G. "QoS-based routing in networks with inacurate information: Theory and algorithms". In: *IEEE Infocom 97*, Kobe, Japan, 1997.

[50] Lorenz, D., and Orda, G. "QoS routing in networks with uncertain parameters". In: *IEEE Infocom 98*, 1998.

[51] Li, B., and Nahrstedt, K. "A control-based middleware framework for quality of service adaptions", *IEEE Journal on Selected Areas in Communications*, setembro de 1997.

[52] Li, B., and Nahrstedt, K. "Dinamic reconfiguration for complex multimidia application". In: *IEEE International Conference on Multimedia Computing and Systems 99*, julho de 1999, v. 1, pp. 165–170.

[53] Cheng, R., and Chang, C. "Design of a fuzzy traffic controler for ATM networks", *IEEE/ACM Transactions on Networking v. 4*, n. 3, junho de 1996, pp. 460–469.

[54] Atlasis, A. F., Loukas, N. H., and Vasilakos, A. V. "The use of learning algorithms in ATM networks call admission control problem: a methodology", *Computer Networks v. 34*, n. 3, setembro de 2000, pp. 341–353.

[55] Pitsillides, A., Stylianou, G., Pattichis, C., Sekercioglu, A., and Vasilakos, A. "Bandwidth allocation for virtual paths (BAVP): Investigation of performance of classical constrained and genetic algorithm based optimisation techniques". In: *Proceedings of the 2000 IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM-00)*, Los Alamitos, março de 2000, IEEE, pp. 1501–1510.

[56] Catania, V., Ficili, G., Palazzo, S., and Panno, D. "A comparative analysis of fuzzy versus conventional policing mechanisms for atm networks", *IEEE/ACM Transactions on Networking v. 4*, n. 3, junho de 1996, pp. 449–459.

[57] Ascia, G., Catania, V., Ficili, G., and Panno, D. "A fuzzy buffer management scheme for ATM and IP networks". In: *IEEE Infocom 2001*, 2001, pp. 1539–1547.

[58] Ghosh, S., Razouqi, Q., Schmacher, H. J., and Celmins, A. "A survey of recent advances in fuzzy logic in telecomunications networks and new challenges", *IEEE Transactions on Fuzzy Systems v. 6*, n. 3, agosto de 1998, pp. 443–447.

[59] Zadeh, L. A. "Fuzzy sets", *Information and Control v. 8*, 1965, pp. 338–353.

[60] Zadeh, L. A. "Fuzzy sets". In: *Readings in Fuzzy Sets for Intelligent Systems*, D. Dubois, H. Prade, and R. R. Yager, Eds. Morgan Kaufmann Publishers, 1993, pp. 35–45. ISBN 1-55860-257-7.

[61] Zadeh, L. A. "Fuzzy algorithms", *Information and Control v. 12*, n. 2, fevereiro de 1968, pp. 94–102.

[62] Zadeh, L. A. "Fuzzy logic and its application to approximate reasoning", *Information Processing v. 74*, 1974, pp. 591–594.

[63] Cox, E., O'Hagan, M., and Zadeh, L. *The fuzzy system handbook.* Academic Press Inc, outubro de 1998.

[64] Cordón, O., Herrera, F., and Peregrín, A. "A practical study on the implementation of fuzzy logic controllers", *The International Journal of Intelligent Control and Systems v. 3*, n. 3, junho de 1999, pp. 49–91.

[65] Holland, J. H. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[66] Herrera, F., and Lozano, M. "Fuzzy genetic algorithms: Issues and models.". Relatório técnico, 1998. Technical Report DECSAI-98116, Dept. of Computer Science and A.I., University of Granada.

[67] Arabshahi, P., Choi, J. J., II, R. J. M., and Caudell, T. P. "Fuzzy parameter adaptation in optimization: Some neural net training examples", *IEEE Computing Science and Engineering v. 3*, n. 1, Spring 1996, pp. 57–65.

[68] von Altrock, C. *Fuzzy Logic and NeuroFuzzy Aplications Explained.* Prentice Hall, Englewood Cliffs - NJ, 1995.

[69] Buja, G. S., and Todesco, F. "Neural network implementation of a fuzzy logic controller", *IEEE Transactions on Ind. Electronic v. 41*, n. 1, dezembro de 1994, pp. 663–665.

[70] DMTF. "Common Information Model (CIM) specification - version 2.2". Distributed Management Task Force. http://www.dtmf.org/spec/cims.html, junho de 1999.

[71] Wang, L. X., and Mendel, J. "Generating fuzzy rules by learning from examples", *IEEE Transactions on Systems, Man, and Cybernetics v. 22*, n. 2, julho de 1992, pp. 1414–1427.

[72] Cox, E. *Fuzzy logic for business and industry.* Charles River Media, outubro de 1995.

[73] Michaelewicz, Z. *Genetic algorithms + data structure = evolution programs.* Springer-Verlag, março de 1996.

[74] Mitchell, M. *An Introduction to Genetic Algorithms.* MIT Press, março de 1996.

[75] De Jong, K. *An Analysis of the Behavior of a class of Genetic Adaptive System.* Tese de Doutorado, University of Michigan, 1975.

[76] Mortensen, J. E. "JFS Fuzzy System". http://www.inet.uni2.dk/ jemor/jfs.htm, 1998.

[77] McCanne, S., and Floyd, S. "ns Network Simulator - Version 2". http://www.isi.edu/nsnam/ns/, 1998.

[78] Pieda, P., Ethridge, J., and Baines, M. "A network simulator differentiated services implementation - Open IP". http://www7.nortel.com:8080/CTL/, outubro de 2000.

[79] Ross, T. *Fuzzy Logic with Engineering Applications.* ISBN 0-07-053917-0. McGraw-Hill, New York, 1985.

[80] Cordón, O., Herrera, F., and Peregrín, A. "Looking for the best defuzzification method features for each implication operator to design accurate fuzzy models". Relatório técnico, University of Granada, abril de 1999. Technical Report DECSAI-99108, Dept. of Computer Science and A.I., University of Granada.

[81] Zegura, E. W., Calvert, K., and Bhattacharjee, S. "How to model an internetwork". In: *IEEE Infocom 96*, San Francisco, USA, março de 1996.

[82] Zegura, E. W., Calvert, K. L., and Donahoo, M. J. "A quantitative comparison of graph-based models for internet topology", *IEEE/ACM Transactions on Networking v. 5*, n. 6, dezembro de 1997, pp. 770–783.

[83] Calvert, K., Doar, M., and Zegura, E. W. "Modeling internet topology", *IEEE Communications Magazine*, junho de 1997.

[84] Waxman, B. M. "Routing of multipoint connections", *IEEE Journal on Selected Areas in Communications v. 6*, n. 9, 1988, pp. 1617–1622.

[85] Hsiung, H., Fischer, M. J., Masi, D. M., Cuffie, D., and Scheurich, S. "An approach to IP telephony performance measurement and modeling in government environments". In: *9th Annual Conference of the Internet Society, INET'99*, San Jose, USA, julho de 1999.

[86] Damianou, N. C. *A Policy Framework for Management of Distributed Systems*. Tese de Doutorado, Imperial College - University of London, 2002.

[87] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. "Ponder: A language for specifying security and management policies for distributed systems". Relatório Técnico Version 2.3, Imperial College of Science, Technology and Medicine - University of London, London, UK, outubro de 2000.

[88] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. "A policy deployment model for the Ponder language". In: *IEEE/IFIP International Symposium on Integrated Network Management*, Seattle, USA, maio de 2001.

[89] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. "Tools for domain-based policy management of distributed systems". In: *IEEE/IFIP Network Operation and Management Sysmposium (NOMS2002)*, Florence, Italy, abril de 2002.

Apêndice A

Ponder: Linguagem para especificação de políticas

A LINGUAGEM *Ponder* [47] foi criada por Damianou *et al.*[48] para especificar textualmente as políticas de gerenciamento propostas por Sloman [34]. A linguagem Ponder é declarativa e orientada a objetos e oferece ao usuário uma interface simples para especificação de políticas

Escolhemos essa linguagem porque nos permitiu especificar diretamente as políticas, tendo em vista que o presente trabalho também foi baseado nas propostas de Sloman [34]. Apresentamos, a seguir, uma introdução à linguagem Ponder, bem como os comandos relacionados a esse trabalho. A especificação completa da linguagem poderá ser obtidas em Damianou[86, 87, 88, 86]. A ferramenta utilizada para especificar e validar as políticas foi o *Ponder Toolkit* [47, 89].

A.1 Sintaxe da linguagem Ponder

Apresentamos, a seguir, a sintaxe da linguagem *Ponder*.

A.1.1 Estrutura de um programa Ponder

A estrutura da linguagem ponder é composta de cinco partes principais, cujas palavras-chaves são apresentadas abaixo:

domain Especifica o domínio de aplicação da política. Uma política tem uma área de atuação, a qual é chamada de domínio[35]. Toda especificação *Ponder* inicia com a identificação do domínio a que se referencia a política. Além disso, também aumenta a clareza da especificação por permitir a atribuição de apelidos.

inst Indica a instância de uma política especificando as características e restrições. Sua declaração equivale a uma instância de um objeto.

type Descreve uma estrutura comum de uma determinada política. Vários elementos com mesmo comportamento não precisam especificá-lo a cada instância, bastando definir um tipo. Sua declaração equivale a especificação de uma classe.

constraint Indica alguma restrição à aplicação de uma política.

event Eventos são condições de disparo das políticas, que podem ser agrupadas para especificar eventos complexos. É conveniente especificar eventos separadamente, para poder reutilizá-los em várias especificações de políticas.

Apresentamos no código A.1 um trecho de especificação em Ponder. Aqui o domínio departamento de *vendas* na *cidade* recebe o apelido *a*. O tipo *iniciaComputador* especifica a política para iniciar um computador qualquer. A instância *iniciaServidor* especifica a política de iniciar o servidor a partir do tipo *iniciaComputador*.

Código A.1: Exemplo de estrutura de uma especificação Ponder

```
domain
    a = /cidade/vendas
constraint
    horario = relogioEntre(0800,1700);
event
    pressionarBotao = botaoReset -> pressionado;
type
    oblig iniciaComputador(subject s, target t) {
        on s.pressionarBotao()
        do t.reset()
        when horario
    }
inst
    oblig iniciaServidor = iniciaComputador(a/supervisor, a/servidor)
```

A.1.2 Políticas básicas

As políticas básicas são apresentadas a seguir:

auth Indica uma política de autorização, isto é, políticas relativas ao controle de acesso e especifica o que um sujeito pode fazer **auth+** ou o que um sujeito não pode fazer **auth-**.

oblig Indica uma política de obrigação. Essa política executada por um sujeito é disparada por um evento e acompanhada de uma ação, obrigatória de ser executada quando ocorre esse evento.

refrain Indica uma política de proibição. Essa política proíbe que um sujeito execute alguma ação, assim, é similar a uma autorização negativa **auth-**, porém interpretada pelo sujeito.

deleg Indica quais ações um sujeito está autorizado a delegar para outros. Corresponde à autorização de poder delegar alguma ação **deleg+** ou de não poder delegar determinada ação **deleg-**.

Apresentamos, no código A.2, um trecho de especificação em Ponder de políticas básicas. Nesse exemplo usamos o comando **inst** por ser uma especificação simples, mas ela poderia também ser definida através do comando **type**. Aqui a política *iniciaComputador* tem uma especificação de autorização **auth+** e uma de obrigação **oblig**.

Código A.2: Exemplo de uma especificação de política básica na linguagem Ponder

```
event
    pressionarBotao = botaoReset -> pressionado;
inst
    auth+ iniciaComputador {
        subject a/supervisor
        target a/estacaoB
        action a/estacaoB.reset()
        when pressionarBotao
    }
inst
    oblig iniciaComputador {
        subject a/estacaoA
        target a/estacaoB
        on pressionarBotao
        do a/estacaoB.reset()
```

```
    }
```

Política de autorização

Uma política de autorização especifica o controle de acesso do sistema, isto é, que ações um determinado sujeito pode executar. Ela pode ser positiva, permitindo o acesso, ou negativa, negando o acesso. Uma política de autorização pode conter os seguintes campos:

subject Indica o sujeito que pode ou não realizar a ação.

target Indica o elemento alvo em que a ação será realizada.

when Indica uma condição de execução.

action Descreve a ação que o sujeito está autorizado a realizar no alvo caso as condições de execução e restrição sejam satisfeitas.

Tanto os comandos **subject** como **target** são apresentados no formato **domain**.

Apresentamos, no código A.3, um trecho de especificação em Ponder de políticas básicas. Nesse exemplo o comando **auth+** indica uma política de autorização positiva (permissão) ao usuário *a/supervisor* de realizar um comando *reset()* no alvo *a/estacaoB* quando ocorrer *pressionaBotao* e satisfizer a restrição de *horario*.

Código A.3: Exemplo de uma especificação de política de autorização em Ponder

```
domain
    a = /cidade/vendas
constraint
    horario = relogioEntre(0800,1700);
event
    pressionarBotao = botaoReset -> pressionado;
inst
    auth+ iniciaComputador {
        subject a/supervisor
        target a/estacaoB
        action a/estacaoB.reset()
        when pressionarBotao and horario
    }
```

Política de obrigação

Uma política de obrigação indica uma ação obrigatória quando ocorre um evento. Ela é indicada através da palavra-chave **oblig**.

subject Indica o sujeito que pode ou não realizar a ação.

target Indica o elemento alvo em que a ação será realizada.

on Indica o evento que dispara a ação.

do Descreve a ação que o sujeito é obrigado a executar no alvo caso o evento ocorra.

catch Indica a ação de exceção, isto é, o que deve fazer caso a execução da ação falhe.

Apresentamos, no código A.4, um trecho de especificação em Ponder de política de obrigação. Nesse exemplo, não usamos o comando **domain** e os nomes são exibidos completos. O comando **oblig** indica uma política de obrigação do elemento *estacaoA* realizar um comando *servidor.reset()* no alvo *servidor*, quando ocorrer *servidorNaoresponde*. Esse evento é definido como execução 30 vezes do comando ping sem resposta. É claro que, para essa política ser executada, é necessária autorização.

Código A.4: Exemplo de uma especificação de política de obrigação em Ponder

```
event
    timeOut = pingSemresposta;
    servidorNaoresponde = 30 * timeOut;
inst
    oblig iniciaComputador {
        subject /cidade/vendas/estacaoA
        target /cidade/vendas/servidor
        on servidorNaoresponde
        do /cidade/vendas/servidor.reset()
        catch acendeAlarme()
    }
```

Política de proibição

Uma política de proibição indica uma ação que o sujeito não pode executar. Ela é indicada através da palavra-chave **refrain**. Essa política é semelhante a uma autorização negativa, sendo que a proibição é implementada pelo sujeito, e pode ser útil quando o alvo não deseja entregar sua proteção a um outro sujeito (como ocorre no caso de autorização negativa).

subject Indica o sujeito que pode ou não realizar a ação.

target Indica o elemento alvo em que a ação será realizada.

action Descreve a ação que o sujeito é proibido de executar no alvo, caso o evento ocorra.

when Indica o evento que dispara a proibição.

Apresentamos, no código A.5, um trecho de especificação em Ponder de política de proibição. O comando **refrain** indica uma política de proibição do usuário *a/estacaoA* realizar um comando *servidor.reset()* no alvo *a/servidor*, quando ocorrer *servidorEmmanutencao*.

Código A.5: Exemplo de uma especificação de política de proibição em Ponder

```
domain a = /cidade/vendas
event
    servidorEmmanutencao = gabineteAberto;
inst
    refrain iniciaComputador {
        subject a/estacaoA
        target a/servidor
        when servidorEmmanutencao
        action servidor.reset()
    }
```

Política de delegação

Uma política de delegação indica se um sujeito pode ou não delegar o poder de executar uma ação de gerenciamento para outro sujeito. Ela é indicada através da palavra-chave **deleg+** ou **deleg-**, respectivamente delegação positiva ou negativa.

subject Indica o sujeito que pode ou não realizar a delegação.

grantee Indica o elemento alvo que pode (ou não) ser delegado.

action Discrimina a ação que o sujeito pode delegar.

when Indica um evento que dispara a delegação.

Apresentamos, no código A.6, um trecho de especificação em Ponder de política de delegação. O comando **deleg+** indica que o usuário *a/gerente* pode delegar ao usuário *a/supervisor* o poder de executar uma ação *servidor.reset()* no alvo *a/servidor*. Observamos que o *servicoManutencao* precisa estar autorizado para *a/gerente*.

Código A.6: Exemplo de uma especificação de política de delegação em Ponder

```
domain a = /cidade/vendas
inst
    auth+ servicoManutencao {
          subject a/gerente
          target a/servidor
          action a/servidor.reset()
    }
    deleg+ delegacaoServico (servicoManutencao) {
          subject a/gerente
          grantee a/supervisor
          action reset()
    }
```

Composição de políticas

Para simplificar a especificação de uma política, podemos agrupar um conjunto de políticas aplicáveis a um mesmo sujeito, alvo e evento. A função **group** é muito útil para especificação de políticas de delegação, que exigem também uma autorização.

group Um grupo é um conjunto de domínios ao qual uma determinada política (ou conjunto de políticas) deve ser aplicado.

Apresentamos, no código A.7, um trecho de especificação em Ponder de política de delegação completa (delegação e autorização), organizada com o comando **group**. O funcionamento é semelhante ao apresentado na seção A.1.2, porém para dois grupos: *deptA* e *deptB*.

Código A.7: Exemplo de uma composição de políticas em Ponder

```
domain a = /cidade/vendas
type
    group delegacaoReinicia (subject b, target c, grantee d)
    inst
        auth+ servicoManutencao {
```

```
            subject b
            target c
            action c.reset()
        }
        deleg+ delegacaoServico (servicoManutencao) {
            subject b
            grantee d
            action reset()
        }
    }
inst
    group deptA = delegacaoReinicia(a/gerente, a/servidor, a/↯
        supervisor)
    group deptB = delegacaoReinicia(a/gerente, a/servidor, a/↯
        supervisor)
```

A.2 Exemplo de programa Ponder

Apresentamos, no código A.8, uma especificação Ponder completa. Ela define as políticas para reiniciar o servidor do departamento de vendas nas cidades A e B.

Código A.8: Código exemplo de uma descrição de política na linguagem Ponder

```
constraint
    horario = relogioEntre(0800,1700);
event
    pressionarBotao = botaoReset -> pressionado;
    timeOut = pingSemresposta;
    servidorNaoresponde = 30 * timeOut;
    servidorEmmanutencao = gabineteAberto;
type
    auth+ iniciaComputador(subject s, target t) {
        subject s ;
        target t ;
        action t.reset() ;
        when pressionarBotao and horario ;
    }
    oblig iniciaComputador(subject s, target t) {
        subject s ;
        target t ;
        on s.pressionarBotao() ;
        do t.reset() ;
        when horario ;
    }
```

```
    oblig reiniciaComputador(subject s, target t) {
        subject s ;
        target t ;
        on t.servidorNaoresponde ;
        do t.reset() ;
        catch s.acendeAlarme() ;
    }
    refrain bloqueiaComputador(subject s, target t) {
        subject s ;
        target t ;
        when t.servidorEmmanutencao ;
        action t.reset() ;
    }

domain
    a = /cidadeA/vendas ;
inst
    auth+ servidorA = iniciaComputador(a/supervisor, a/servidor) ;
    oblig servidorA = iniciaComputador(a/supervisor, a/servidor) ;
    refrain servidorA = bloqueiaComputador(a/supervisor, a/servidor) ;

domain
    b = /cidadeB/vendas ;
inst
    auth+ servidorB = iniciaComputador(b/supervisor, b/servidor) ;
    oblig servidorB = iniciaComputador(b/supervisor, b/servidor) ;
    refrain servidorB = bloqueiaComputador(b/supervisor, b/servidor) ;
```

Apêndice B

Simulador de redes *ns*

O *NS* (*Network Simulator*) [77] é um simulador de serviços e de protocolos de rede, que vem sendo utilizado principalmente para a *Internet*. Esse simulador encontra-se em desenvolvimento dentro do projeto *Virtual InterNet Testbed* (VINT), em regime de colaboração entre a Universidade da Califórnia em Berkeley, o *Lawrence Berkeley National Laboratory* (LBL), o *Information Sciences Institute* (ISI) da Universidade da Califórnia do Sul (USC) e o laboratório Xerox PARC.

O *ns* utiliza as linguagens C++ e OTcl (*Object Tool Command Language*), sendo que o seu núcleo é implementado em C++, para permitir melhor desempenho. Para serem executadas, as simulações são configuradas através de *scripts* OTcl, nos quais são descritos, no mínimo, a topologia, o protocolo e as aplicações a serem simuladas.

As topologias são criadas no *ns* através de nós e da conexão desses nós, realizadas por enlaces com características específicas de banda passante e de atraso. As topologias podem ser geradas manualmente ou através do gerador de topologias GT-ITM (*Georgia Tech - Internetwork Topology Models*).

A estrutura dos nós no *ns* é apresentada na figura B.1. Essa estrutura é composta de agentes, um ponto de entrada no nó, um classificador de endereços e um classificador de portas. Os agentes são entidades produtoras ou consumidoras de pacotes e implementam determinados tipos de protocolos. Um pacote gerado por um agente é entregue ao nó ao qual o agente está ligado, através do ponto de entrada, que também recebe pacotes cujo destino é o próprio nó. Após passar pelo ponto de entrada, o pacote é recebido pelo classificador de endereços, que verifica se o pacote deve ser entregue a um agente pertencente ao nó ou deve ser transmitido para um enlace de saída. Caso o pacote seja destinado a um agente do próprio nó, é então repassado ao classificador de porta que, de acordo com o endereço de destino, o entrega ao respectivo agente.

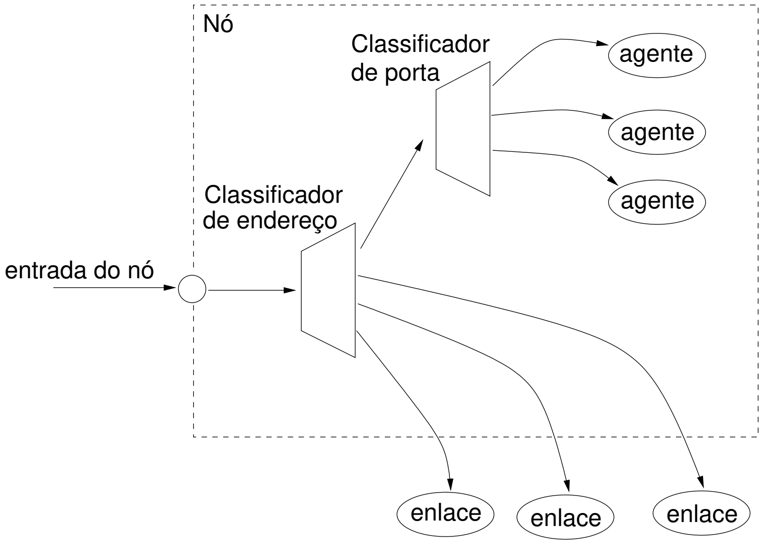


Figura B.1: A estrutura dos nós no *ns*.

Existem vários agentes já implementados no *ns*, tais como o UDP e diversos tipos de TCP (Reno, Vega, NewReno, Tahoe e outros). Nesse trabalho, foram utilizados os agentes *UDP*, *CBR* e *On-Off Exponencial*.

B.1 Gerador de topologias GT-ITM

O gerador de topologias GT-ITM (*Georgia Tech - Internetwork Topology Model*), criado no *College of Computing* do *Georgia Institute of Technology*, é utilizado para a geração de diversos tipos de topologias complexas. Essas topologias são baseadas em um modelo padrão de geração de grafos, que distribui os vértices em posições aleatórias de um plano, havendo uma probabilidade p de uma aresta ser adicionada entre um par de vértices. Tal modelo de geração de grafos, se aplicado sozinho, não reflete a estrutura das redes reais, sendo por isso, propostos outros modelos que alteram a função de probabilidade de um vértice para representarem de forma mais fiel a estrutura dessas redes [82]. Em relação a esses novos modelos, o GT-ITM pode gerar topologias planas, hierárquicas de nível n e *transit-stub*.

O código B.1, mostrado a seguir, apresenta os dados utilizados para a geração das topologias simuladas. A primeira linha após os comentários indica, respectivamente, o tipo de grafo a ser gerado, o número de grafos e a semente a ser utilizada para a geração dos números aleatórios. A linha *4* significa que existirá somente um domínio randômico plano. A linha *5* significa que o domínio deverá ter 30 nós, distribuídos em uma matriz de dimensão 30 (30x30). O parâmetro seguinte indica o método para geração das arestas (neste exemplo, Waxman 2) e a probabilidade de dois nós estarem conectados (19%) e, finalmente, o último parâmetro indica a relação de probabilidade de conexão, de acordo com a distância euclideana (a probabilidade da conexão de dois pontos próximos é maior do que a de pontos distantes). O exemplo abaixo representa uma topologia usada nas simulações e as demais topologias foram criadas conforme os parâmetros especificados na tabela 5.1.

Código B.1: Código para criação de topologias aleatórias no GT-ITM

```
# <method keyword> <number of graphs> [<initial seed>]
# <n> <scale> <edgemethod> <alpha> [<beta>] [<gamma>]

geo 1
30 30 2 0.19 10
```

Apêndice C

Ferramenta para desenvolvimento de controlador fuzzy (JFS)

O JFS [76] é um ambiente de programação para desenvolvimento e execução de programas, com funções de lógica fuzzy e técnicas de aprendizado. Foi a ferramenta escolhida para construir o controlador fuzzy utilizado nesse trabalho.

C.1 Arquitetura do sistema JFS

Um sistema fuzzy é executado a partir de um programa-jfs. O programa-jfs é criado a partir do arquivo de entrada chamado arquivo-jfs, escrito com um editor de texto qualquer. Uma vez escrito, o arquivo-jfs pode ser (a) compilado para uma forma intermediária chamada "p-code"e executado por um dos programas do ambiente; (b) convertido para Javascript e executado em um navegador; (c) executado a partir de outros programas como uma biblioteca dll; (d) executado de um programa em C que usa sua biblioteca; ou (e) convertido em código fonte C a fim de ser incluído em programas C do usuário. O ambiente de desenvolvimento JFS inclui ferramentas para compilar, executar, depurar, otimizar e converter arquivo-jfs em código C.

O sistema JFS combina elementos de linguagens procedurais tradicionais como Pascal, C ou BASIC, com lógica nebulosa. A grande vantagem disso é que o sistema não precisa ser representado apenas com variáveis e operadores fuzzy, podendo incluir funções procedurais na sua especificação. O código fonte (arquivo-jfs) se parece muito com um programa escrito em Pascal. O JFS, no entanto, não tem nenhuma função interna para controlar strings, arquivos, janelas etc. Não é um substituto para linguagens de propósito geral como C++, BASIC, ou Java, mas um programa jfs pode ser chamado de programas escritos nessas outras linguagens.

O arquivo-jfs é escrito a partir de um editor de texto (vi, emacs etc). O compilador, simulador, otimizador e demais ferramentas são executados a partir da linha de comando. Mostramos, na figura C.1, o diagrama dos programas e arquivos que compõem o sistema JFS.

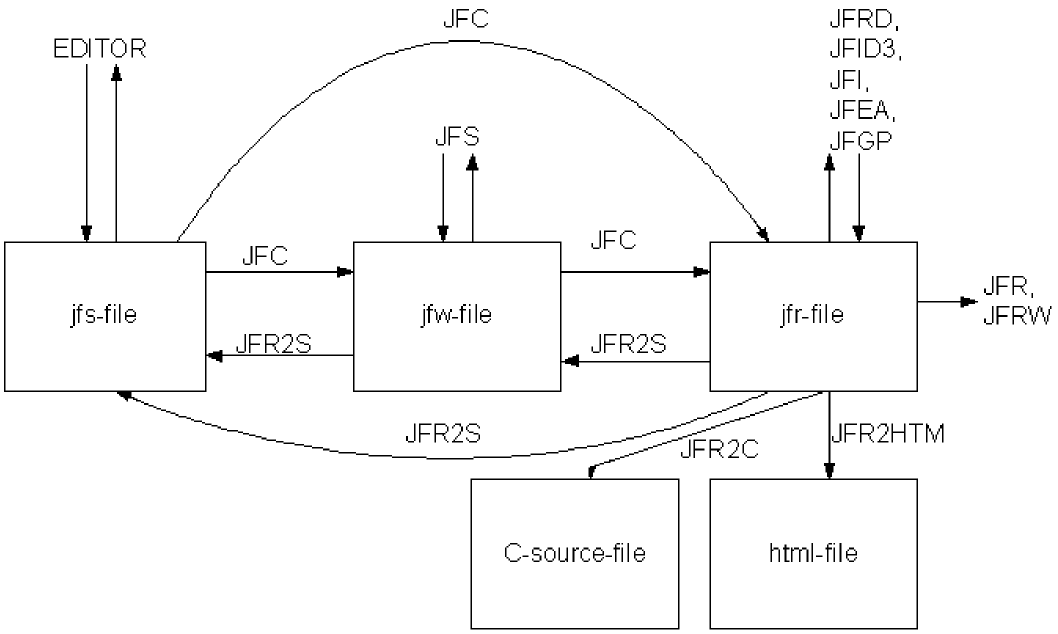


Figura C.1: Arquitetura do sistema JFS.

C.2 Usando o sistema JFS

Um programa-jfs pode ser armazenado em 3 formatos de arquivo: arquivo-jfs, arquivo-jfw e arquivo-jfr. Um arquivo-jfs é um arquivo com código fonte ASCII, que pode ser editado e impresso usando um editor de texto normal (Vi, por exemplo). Um arquivo-jfr é um arquivo em "p-code" binário, que pode ser interpretado (executado) pelo programa JFR. Um arquivo-jfw é um arquivo intermediário entre o jfs e o formato jfr, isto é, com definições em forma binária e o bloco de programa em formato de texto.

O processo de compilar um programa-jfs consiste em converter um arquivo-jfs em um arquivo-jfr. Isso é realizado em um processo de dois passos: primeiro, o arquivo-jfs é convertido a um arquivo-jfw e depois, o arquivo-jfw, é convertido a um arquivo-jfr.

Vantagens do sistema JFS são: (a) um programa compilado pode ser alterado (melhorado) e (b) um programa compilado pode ser reconvertido em código fonte (arquivo-jfs). Essa conversão posterior é chamada compilação inversa e é também um processo de dois passos: primeiro, o arquivo-jfr é convertido a um arquivo-jfw, que, por sua vez, é convertido a um arquivo-jfs.

A partir do arquivo-jfr, pode-se aplicar mecanismos de otimização. Nesse trabalho, usamos o algoritmo Wang-Mendel, conforme descrito na seção 3.5.3, para criação de um conjunto de regras coerentes e um algoritmo genético, descrito na seção 3.7, para otimização dos parâmetros das funções de pertinência.

C.3 Exemplo de utilização do sistema JFS

Um arquivo-jfs se parece muito com um programa escrito em uma linguagem de programação tradicional, como C ou Pascal. Contém o mesmo tipo de declarações e comandos. Abaixo, mostramos, no código C.1, o programa-jfs inicial do controlador do condicionador.

Código C.1: Código do controlador do condicionador inicial

```
title "Conditioner Controller";

operators
    and type min;
    or type max;
    imp type min;
    bunion type max;

domains
    bucket_in type float "bps" 0.0 1.0;
    delay type float "msec" 0.0 1.0;
    drops type float "pkts/s" 0.0 1.0;
    bucket_out type float "bps" 0.0 1.0;

```

```
input
    rate_in "bucket_rate_in" domain bucket_in;
    ef_delay "ef_delay" domain delay;
    ef_discard "ef_discard" domain drops;

output
    rate_out "bucket_rate_out" domain bucket_out defuz centroid;

adjectives
    rate_in MinLowRate center %0.0 base %0.2;
    rate_in VeryLowRate center %0.2 base %0.2;
    rate_in LowRate center %0.35 base %0.2;
    rate_in MediumRate center %0.5 base %0.2;
    rate_in HighRate center %0.65 base %0.2;
    rate_in VeryHighRate center %0.8 base %0.2;
    rate_in MaxHighRate center %1.0 %0.2;

    ef_delay LowDelay 0:1 %0.1:1 %0.3:0;
    ef_delay MediumDelay center %0.5 base %0.6;
    ef_delay HighDelay %0.7:0 %0.9:1 1.0:1;

    ef_discard LowDiscard 0:1 %0.1:1 %0.3:0;
    ef_discard MediumDiscard center %0.5 base %0.6;
    ef_discard HighDiscard %0.7:0 %0.9:1 1.0:1;

    rate_out MinLowRate center 0.0 base %0.2;
    rate_out VeryLowRate center %0.2 base %0.2;
    rate_out LowRate center %0.35 base %0.2;
    rate_out MediumRate center %0.5 base %0.2;
    rate_out HighRate center %0.65 base %0.2;
    rate_out VeryHighRate center %0.8 base %0.2;
    rate_out MaxHighRate center 1.0 base %0.2;

program
extern jfrd input rate_in ef_delay ef_discard
            output rate_out;
```

Note-se, nesse exemplo, que não há base de regras, mas apenas a referência para a utilização do algoritmo Wang-Mendel nas últimas linhas. O arquivo de dados, usado como referência para criação das regras, é mostrado no código C.2.

Código C.2: Arquivo com base de dados para otimização

```
# TB_rate_in    ef_core_delay    ef_discard     TB_rate_out

MinLowRate      LowDelay         LowDiscard     MinLowRate
```

```
VeryLowRate     LowDelay      LowDiscard      VeryLowRate
LowRate         LowDelay      LowDiscard      LowRate
MediumRate      LowDelay      LowDiscard      MediumRate
HighRate        LowDelay      LowDiscard      HighRate
VeryHighRate    LowDelay      LowDiscard      VeryHighRate
MaxHighRate     LowDelay      LowDiscard      MaxHighRate

MinLowRate      LowDelay      MediumDiscard   VeryLowRate
VeryLowRate     LowDelay      MediumDiscard   LowRate
LowRate         LowDelay      MediumDiscard   MediumRate
MediumRate      LowDelay      MediumDiscard   HighRate
HighRate        LowDelay      MediumDiscard   VeryHighRate
VeryHighRate    LowDelay      MediumDiscard   MaxHighRate
MaxHighRate     LowDelay      MediumDiscard   MaxHighRate

MinLowRate      LowDelay      HighDiscard     LowRate
VeryLowRate     LowDelay      HighDiscard     MediumRate
LowRate         LowDelay      HighDiscard     HighRate
MediumRate      LowDelay      HighDiscard     VeryHighRate
HighRate        LowDelay      HighDiscard     MaxHighRate
VeryHighRate    LowDelay      HighDiscard     MaxHighRate
MaxHighRate     LowDelay      HighDiscard     MaxHighRate

MinLowRate      MediumDelay   LowDiscard      MinLowRate
VeryLowRate     MediumDelay   LowDiscard      MinLowRate
LowRate         MediumDelay   LowDiscard      VeryLowRate
MediumRate      MediumDelay   LowDiscard      LowRate
HighRate        MediumDelay   LowDiscard      MediumRate
VeryHighRate    MediumDelay   LowDiscard      HighRate
MaxHighRate     MediumDelay   LowDiscard      VeryHighRate

MinLowRate      MediumDelay   MediumDiscard   MinLowRate
VeryLowRate     MediumDelay   MediumDiscard   MinLowRate
LowRate         MediumDelay   MediumDiscard   VeryLowRate
MediumRate      MediumDelay   MediumDiscard   LowRate
HighRate        MediumDelay   MediumDiscard   MediumRate
VeryHighRate    MediumDelay   MediumDiscard   HighRate
MaxHighRate     MediumDelay   MediumDiscard   VeryHighRate

MinLowRate      MediumDelay   HighDiscard     MinLowRate
VeryLowRate     MediumDelay   HighDiscard     MinLowRate
LowRate         MediumDelay   HighDiscard     VeryLowRate
MediumRate      MediumDelay   HighDiscard     LowRate
HighRate        MediumDelay   HighDiscard     MediumRate
VeryHighRate    MediumDelay   HighDiscard     HighRate
MaxHighRate     MediumDelay   HighDiscard     VeryHighRate
```

```

MinLowRate      HighDelay     LowDiscard      MinLowRate
VeryLowRate     HighDelay     LowDiscard      MinLowRate
LowRate         HighDelay     LowDiscard      MinLowRate
MediumRate      HighDelay     LowDiscard      VeryLowRate
HighRate        HighDelay     LowDiscard      LowRate
VeryHighRate    HighDelay     LowDiscard      MediumRate
MaxHighRate     HighDelay     LowDiscard      HighRate

MinLowRate      HighDelay     MediumDiscard   MinLowRate
VeryLowRate     HighDelay     MediumDiscard   MinLowRate
LowRate         HighDelay     MediumDiscard   MinLowRate
MediumRate      HighDelay     MediumDiscard   VeryLowRate
HighRate        HighDelay     MediumDiscard   LowRate
VeryHighRate    HighDelay     MediumDiscard   MediumRate
MaxHighRate     HighDelay     MediumDiscard   HighRate

MinLowRate      HighDelay     HighDiscard     MinLowRate
VeryLowRate     HighDelay     HighDiscard     MinLowRate
LowRate         HighDelay     HighDiscard     MinLowRate
MediumRate      HighDelay     HighDiscard     VeryLowRate
HighRate        HighDelay     HighDiscard     LowRate
VeryHighRate    HighDelay     HighDiscard     MediumRate
MaxHighRate     HighDelay     HighDiscard     HighRate
```

Após a criação de regras com o algoritmo de Wang-Mendel, obtemos a listagem apresentada no código C.3.

Código C.3: Código do controlador com regras criadas pelo algoritmo de Wang-Mendel

```
title "Conditioner Controller";

operators
    and type min;
    or type max;
    imp type min;
    bunion type max;

domains
    bucket_in "bps" 0 1;
    delay "msec" 0 1;
    drops "pkts/s" 0 1;
    bucket_out "bps" 0 1;

```

```
input
    rate_in "bucket_rate_in" bucket_in;
    ef_delay "ef_delay" delay;
    ef_discard "ef_discard" drops;

output
    rate_out "bucket_rate_out" bucket_out;

adjectives
    rate_in MinLowRate center %0.0 base %0.2;
    rate_in VeryLowRate center %0.2 base %0.2;
    rate_in LowRate center %0.35 base %0.2;
    rate_in MediumRate center %0.5 base %0.2;
    rate_in HighRate center %0.65 base %0.2;
    rate_in VeryHighRate center %0.8 base %0.2;
    rate_in MaxHighRate center %1.0 %0.2;

    ef_delay LowDelay 0:1 %0.1:1 %0.3:0;
    ef_delay MediumDelay center %0.5 base %0.6;
    ef_delay HighDelay %0.7:0 %0.9:1 1.0:1;

    ef_discard LowDiscard 0:1 %0.1:1 %0.3:0;
    ef_discard MediumDiscard center %0.5 base %0.6;
    ef_discard HighDiscard %0.7:0 %0.9:1 1.0:1;

    rate_out MinLowRate center 0.0 base %0.2;
    rate_out VeryLowRate center %0.2 base %0.2;
    rate_out LowRate center %0.35 base %0.2;
    rate_out MediumRate center %0.5 base %0.2;
    rate_out HighRate center %0.65 base %0.2;
    rate_out VeryHighRate center %0.8 base %0.2;
    rate_out MaxHighRate center 1.0 base %0.2;

program
    switch;
    case ef_delay LowDelay;
      switch;
      case ef_discard LowDiscard;
        ifw %0.7 rate_in VeryLowRate then rate_out VeryLowRate;
        ifw %0.7 rate_in LowRate then rate_out LowRate;
        ifw %0.7 rate_in MediumRate then rate_out MediumRate;
        ifw %0.7 rate_in HighRate then rate_out HighRate;
        ifw %0.7 rate_in VeryHighRate then rate_out VeryHighRate;
      end;
      switch;
      case ef_discard MediumDiscard;
```

```
        ifw %0.7 rate_in MinLowRate then rate_out VeryLowRate;
        ifw %0.7 rate_in VeryLowRate then rate_out LowRate;
        ifw %0.7 rate_in LowRate then rate_out MediumRate;
        ifw %0.7 rate_in MediumRate then rate_out HighRate;
        ifw %0.7 rate_in HighRate then rate_out VeryHighRate;
      end;
      switch;
      case ef_discard HighDiscard;
        ifw %0.7 rate_in MinLowRate then rate_out LowRate;
        ifw %0.7 rate_in VeryLowRate then rate_out MediumRate;
        ifw %0.7 rate_in LowRate then rate_out HighRate;
        ifw %0.7 rate_in MediumRate then rate_out VeryHighRate;
        ifw %0.7 rate_in between HighRate and MaxHighRate then rate_out MaxHighRate;
      end;
      ifw %0.7 rate_in MaxHighRate then rate_out MaxHighRate;
      ifw %0.7 rate_in between VeryHighRate and MaxHighRate and ef_discard between MediumDiscard and HighDiscard then rate_out MaxHighRate;
    end;
    switch;
    case ef_delay HighDelay;
      ifw %0.7 rate_in between MinLowRate and LowRate then rate_out MinLowRate;
      ifw %0.7 rate_in MediumRate then rate_out VeryLowRate;
      ifw %0.7 rate_in HighRate then rate_out LowRate;
      ifw %0.7 rate_in VeryHighRate then rate_out MediumRate;
      ifw %0.7 rate_in MaxHighRate then rate_out HighRate;
    end;
    switch;
    case ef_delay MediumDelay;
      ifw %0.7 rate_in LowRate then rate_out VeryLowRate;
      ifw %0.7 rate_in MediumRate then rate_out LowRate;
      ifw %0.7 rate_in HighRate then rate_out MediumRate;
      ifw %0.7 rate_in VeryHighRate then rate_out HighRate;
      ifw %0.7 rate_in MaxHighRate then rate_out VeryHighRate;
    end;
    ifw %0.7 rate_in MinLowRate and ef_discard LowDiscard then rate_out MinLowRate;
    ifw %0.7 rate_in between MinLowRate and VeryLowRate and ef_delay between MediumDelay and HighDelay then rate_out MinLowRate;
```

A última etapa é a otimização dos parâmetros do controlador fuzzy, com o algoritmo genético. Observe-se que, nas definições das funções de pertinência, alguns valores apresentam o caracter % adiante das variáveis que desejamos otimizar. Após a otimização, obtemos o código C.4, mostrado a seguir.

Código C.4: Código do controlador com parâmetros otimizados pelo algoritmo genético

```
title "Conditioner Controller";

operators
    and type min;
    or type max;
    imp type min;
    bunion type max;

domains
    bucket_in "bps" 0 1;
    delay "msec" 0 1;
    drops "pkts/s" 0 1;
    bucket_out "bps" 0 1;

input
    rate_in "bucket_rate_in" bucket_in;
    ef_delay "ef_delay" delay;
    ef_discard "ef_discard" drops;

output
    rate_out "bucket_rate_out" bucket_out;

adjectives
    rate_in MinLowRate %0.1748;
    rate_in VeryLowRate %0.3086 base %0.3555;
    rate_in LowRate %0.4609 base %0.0566;
    rate_in MediumRate %0.5914 base %0.9629;
    rate_in HighRate %0.695 base %0.3746;
    rate_in VeryHighRate %0.847 base %0.2772;
    rate_in MaxHighRate %0.9708;

    ef_delay LowDelay 0:1 %0.0054:1 %0.0224:0.0107;
    ef_delay MediumDelay %0.0363 base %0.7262;
    ef_delay HighDelay %0.7816:0 %0.836:0.9957 1:1;

    ef_discard LowDiscard 0:1 %0.2295:1 %0.3835:0.0003;
    ef_discard MediumDiscard %0.4974 base %0.0111;
    ef_discard HighDiscard %0.7098:0 %0.8354:0.894 1:1;

    rate_out MinLowRate 0 base %0.2632;
    rate_out VeryLowRate %0.257 base %0.4378;
```

```
    rate_out LowRate %0.4323 base %0.4482;
    rate_out MediumRate %0.5559 base %0.209;
    rate_out HighRate %0.6752 base %0.1721;
    rate_out VeryHighRate %0.7877 base %0.1223;
    rate_out MaxHighRate 1 base %0.0237;

program
    switch;
    case ef_delay LowDelay;
      switch;
      case ef_discard LowDiscard;
        ifw %0.3956 rate_in VeryLowRate then rate_out VeryLowRate;
        ifw %0.4159 rate_in LowRate then rate_out LowRate;
        ifw %0.5425 rate_in MediumRate then rate_out MediumRate;
        ifw %0.2048 rate_in HighRate then rate_out HighRate;
        ifw %0.3955 rate_in VeryHighRate then rate_out VeryHighRate;
      end;
      switch;
      case ef_discard MediumDiscard;
        ifw %0.7686 rate_in MinLowRate then rate_out VeryLowRate;
        ifw %0.4809 rate_in VeryLowRate then rate_out LowRate;
        ifw %0.6260 rate_in LowRate then rate_out MediumRate;
        ifw %0.4600 rate_in MediumRate then rate_out HighRate;
        ifw %0.3429 rate_in HighRate then rate_out VeryHighRate;
      end;
      switch;
      case ef_discard HighDiscard;
        ifw %0.1134 rate_in MinLowRate then rate_out LowRate;
        ifw %0.1342 rate_in VeryLowRate then rate_out MediumRate;
        ifw %0.9088 rate_in LowRate then rate_out HighRate;
        ifw %0.7438 rate_in MediumRate then rate_out VeryHighRate;
        ifw %0.7363 rate_in between HighRate and MaxHighRate then ↯
           rate_out
          MaxHighRate;
      end;
      ifw %0.5190 rate_in MaxHighRate then rate_out MaxHighRate;
      ifw %0.6957 rate_in between VeryHighRate and MaxHighRate and ↯
         ef_discard between MediumDiscard and HighDiscard then ↯
         rate_out MaxHighRate;
    end;
    switch;
    case ef_delay HighDelay;
      ifw %0.3204 rate_in between MinLowRate and LowRate then rate_out
        MinLowRate;
      ifw %0.1884 rate_in MediumRate then rate_out VeryLowRate;
      ifw %0.6055 rate_in HighRate then rate_out LowRate;
```

```
      ifw %0.3274 rate_in VeryHighRate then rate_out MediumRate;
      ifw %0.4854 rate_in MaxHighRate then rate_out HighRate;
    end;
    switch;
    case ef_delay MediumDelay;
      ifw %0.7898 rate_in LowRate then rate_out VeryLowRate;
      ifw %0.4639 rate_in MediumRate then rate_out LowRate;
      ifw %0.5527 rate_in HighRate then rate_out MediumRate;
      ifw %0.4851 rate_in VeryHighRate then rate_out HighRate;
      ifw %0.4673 rate_in MaxHighRate then rate_out VeryHighRate;
    end;
    ifw %0.6452 rate_in MinLowRate and ef_discard LowDiscard then rate_out
      MinLowRate;
    ifw %0.5598 rate_in between MinLowRate and VeryLowRate and ef_delay between MediumDelay and HighDelay then rate_out MinLowRate;
```

Apêndice D

Algoritmos genéticos

O ALGORITMO Genético (AG) é um mecanismo de busca baseado no processo de evolução natural de Charles Darwin e nos mecanismos de genética. Os AGs foram inicialmente propostos por John Holland em 1975[65] e receberam muitas melhorias ao longo das décadas de 80 e 90, acompanhando o aumento da capacidade computacional. Uma aplicação para a qual AGs têm se mostrado uma ferramenta valiosa é a otimização.

A otimização de um controlador fuzzy é um processo que busca encontrar a melhor combinação de parâmetros para atingir o melhor resultado possível. Como os parâmetros (funções de pertinência) de um controlador fuzzy são inter-relacionados, a técnica de AG se mostra bastante útil, pois realiza uma heurística em todo o universo e produz resultados em razoável tempo de processamento.

Apesar dos parâmetros serem aleatórios, a busca é direcionada, tendo como referência as informações históricas para encontrar novos pontos de busca onde são esperados melhores resultados. Isto é feito através de processos iterativos, onde cada iteração é chamada de *geração*.

AGs são algoritmos de otimização global, baseados nos mecanismos de genética e seleção natural. Eles empregam uma estratégia de busca paralela e estruturada em direção da busca de pontos de "alta aptidão", ou seja, pontos nos quais a função a ser otimizada tem valores comparativamente melhores.

D.1 Definições

As principais definições relacionadas com os AGs são:

Gen ou Gene É a unidade básica do cromossomo. Cada gene descreve um certo parâmetro codificado no cromossomo, isto é, um elemento do vetor que representa o cromossomo.

Cromossomo Uma cadeia de genes representando alguma informação relativa às variáveis do problema. Um cromossomo é a estrutura de dados que codifica uma solução possível para um problema, isto é, representa um simples ponto no espaço de busca. Também pode ser chamado de *genótipo*.

Aptidão É o grau de proximidade da solução de um determinado cromossomo em relação à função objetivo. Também pode ser chamado de *fenótipo*

Indivíduo Um membro da população, formado por um cromossomo e a sua aptidão.

População Conjunto de indivíduos que fazem parte do espaço de busca.

Geração Uma população associada a uma iteração que o AG executa.

Operações Genéticas Operações que o AG realiza sobre cada um dos cromossomos.

Espaço de Busca É o conjunto, espaço ou região que compreende as soluções possíveis ou viáveis do problema a ser otimizado. O espaço de busca pode ser muito grande ou até infinito e, por isso, deve ser caracterizado por funções de restrição.

Função Objetivo É a função de referência para a otimização. Ela contém a informação numérica do desempenho desejado para cada cromossomo da população. Uma função é normalmente expressa como: $J = f(x_1, x_2, \ldots, x_n)$, onde $x_1, x_2, \ldots, x_n$ são as variáveis que o algoritmo procura determinar para otimizar J.

D.2 Funcionamento dos algoritmos genéticos

Existem muitos métodos de funcionamento de AGs, com variação de procedimentos e operadores empregados. Entretanto, as mais comuns são: geracional, "em regime" (*steady-state*) e simples.

No método *geracional*, toda a população é substituída por novos indivíduos gerados pelo processo de seleção após a aplicação dos operadores genéticos. Como nesse método todos os "pais" são descartados a cada geração, poderá haver perda de bons indivíduos, retardando a convergência da solução se a mesma estiver próxima.

O método *em regime* tem funcionamento oposto ao método *geracional*. Nesse método, apenas um indivíduo é criado a cada vez. Este, após avaliação, poderá ser inserido na população em substituição a algum outro elemento, por exemplo, o pior deles. Caso seja pior que todos, ele é descartado e procede-se à criação de um novo indivíduo. Com esse método, os melhores indivíduos sempre farão parte da população, entretanto a convergência pode ser lenta se a solução estiver distante do ponto de partida.

O método *simples* tem funcionamento intermediário aos dois métodos anteriores. Sua diferença é que tanto pais como filhos são avaliados em paralelo, possibilitando a escolha dos melhores indivíduos em qualquer situação. Neste caso, obtemos uma convergência rápida para o caso da solução estar próxima ou distante do ponto de partida.O algoritmo genético simples é apresentado no algoritmo D.5.

Algoritmo D.5 Algoritmo genético simples

Geração $t \leftarrow 0$
Inicia a população de cromossomos
Avalia indivíduos na população
enquanto objetivo final não for atendido ou critério de parada atingido **faça**
  Selecionar indivíduos para reprodução
  Aplicar operadores genéticos (cruzamento e/ou mutação)
  Avaliar indivíduos gerados
  Selecionar indivíduos para próxima geração
  Incrementa geração $t \leftarrow t + 1$
fim enquanto

D.3 Representação e codificação

A primeira etapa da aplicação de algoritmo genético consiste em representar genotipicamente o conjunto de parâmetros. Cada gene representa um parâmetro e pode ser um valor binário (0 ou 1), um valor inteiro ou um valor real. O genótipo é um vetor de tamanho finito e, geralmente, com um alfabeto (valores possíveis) finito. Usualmente, representamos indivíduos com um vetor binário por simplicidade, porém no caso de utilização de AG para otimização é usual a utilização de genes inteiros ou reais com casas decimais fixas. Um genótipo produz um resultado, que chamamos fenótipo (características). Associando um genótipo a um fenótipo obtemos um indivíduo.

D.4 Avaliação da população

Durante cada iteração, os operadores de seleção e reprodução são aplicados a uma população (conjunto de indivíduos), na qual a quantidade de indivíduos pode variar em função da complexidade do problema e da capacidade de processamento disponível. Por meio da seleção escolhemos os melhores indivíduos que irão se reproduzir, gerando descendentes para a próxima geração, com uma probabilidade determinada pelo índice de aptidão. Assim os indivíduos mais aptos têm mais chance de se reproduzir.

A avaliação da população é realizada pela função de aptidão, que deve indicar a qualidade de cada indivíduo da população. Para problemas de otimização, esta função está intimamente ligada à função objetivo que se deseja atingir. Para problemas de maximização, a função de aptidão pode ser derivada a partir da função objetivo, conforme mostrado na equação D.1 onde $f(x)$ é a função de aptidão, $g(x)$ é a função objetivo, $r(x)$ é o valor da função que queremos avaliar e k é uma constante introduzida para tornar o valor da função de aptidão mais fácil de trabalhar. Quanto maior for $f(x)$, maior é a proximidade com o valor desejado e a aptidão do valor avaliado.

$$f(x) = \frac{k}{|r(x) - g(x)|} \tag{D.1}$$

D.5 Operações genéticas

A reprodução de indivíduos de uma população se dá através de operadores genéticos: seleção, cruzamento e mutação. Essas operações simulam as operações biológicas para criar uma população de indivíduos cada vez mais aptos. Apresentamos a seguir esses operadores.

D.5.1 Seleção

O principal operador de algoritmo genético é o critério de seleção para possibilitar que a reprodução dos indivíduos produzam indivíduos mais aptos. A maioria dos métodos de seleção escolhem preferencialmente indivíduos com maiores valores de aptidão, mas incluem indivíduos menos aptos para manter a diversidade da população. Um método de seleção muito utilizado é o Método da Roleta, inclusive nesse trabalho.

O Método da Roleta foi proposto por Holland[65] e consiste em ponderar cada indivíduo da população pela sua aptidão. Na equação D.2, mostramos uma forma de calcular a probabilidade p_i do i-ésimo indivíduo da população vir a ser selecionado para reprodução, que é proporcional ao seu valor da função de aptidão $f_i = F(x_i)$. Assumimos que N é o tamanho da população e que f_i seja positiva.

$$p_i = \frac{f_i}{\sum\limits_{j=i}^{N} f_j} \tag{D.2}$$

No Método da Roleta, cada indivíduo da população é representado como um segmento de círculo, no qual o tamanho do setor é proporcional à sua aptidão, conforme mostrado na figura D.1.

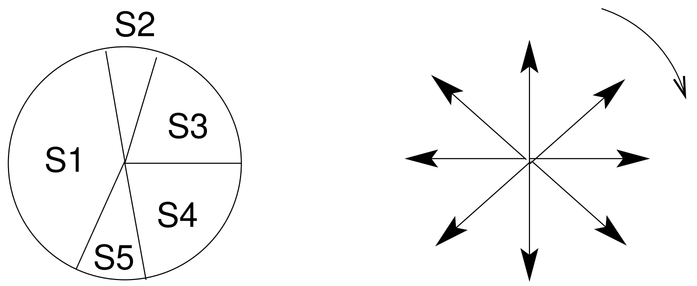


Figura D.1: Diagrama de uma operação de seleção através do método da roleta.

O eixo da roleta é representada por uma quantidade de vetores igual ao tamanho da população desejada e ângulos equidistantes entre vetores. Girando aleatoriamente a roleta, os vetores apontam para os indivíduos selecionados para a próxima geração. Os indivíduos com melhor aptidão terão mais chances de ser escolhidos, dada a maior porção da roleta. A tabela D.1 mostra os valores de aptidão calculados para construção da roleta.

O método da roleta tem a desvantagem de poder levar a um grande número

Tabela D.1: Valores de aptidões para seleção através do método da roleta.

	Indivíduo S_i	Aptidão $f(S_i)$	Aptidão Relativa
S_1	001011	14,4	48 %
S_2	010010	2,1	7 %
S_3	011011	3,9	13 %
S_4	110011	6,0	20 %
S_5	010001	3,6	12 %

de cópias de um bom cromossomo, diminuindo a diversidade da população. A baixa diversidade pode provocar uma convergência prematura para uma solução máxima local, o que não é desejado. A vantagem desse método é a baixa necessidade computacional e o tempo curto de processamento em relação aos outros métodos. Porém, utilizando uma taxa de mutação maior (que insere mais indivíduos aleatórios na população) podemos minorar essa deficiência, aproveitando o bom desempenho.

Outro método de seleção é a Seleção por Ordenação, onde os indivíduos são ordenados por suas aptidões e a seleção se dá através de uma função de probabilidade decrescente linear ou exponencial. Os indivíduos mais aptos, portanto, têm maior probabilidade. Uma escolha cuidadosa dessa função possibilita manter uma boa diversidade e convergência rápida, porém a carga computacional é alta, pois é necessário ordenar todos os indivíduos a cada geração.

A Seleção por Torneio consiste em promover um torneio entre N indivíduos ($N > 2$), aleatoriamente escolhidos na população. O melhor indivíduo é selecionado e os demais são descartados. Nesse método o esforço computacional é pequeno e não há problema da convergência prematura. Pelo contrário, a convergência pode até ser excessivamente lenta, já que alguns bons indivíduos serão descartados, caso disputem com outros melhores.

D.5.2 Cruzamento

O cruzamento é o operador responsável pela recombinação de características dos pais durante a reprodução, permitindo que as próximas gerações herdem essas características. Ele é considerado o operador genético predominante, por isso a probabilidade dada pela taxa de cruzamento P_C, deve ser maior que a taxa de mutação. Existem várias formas de aplicar o operador de acordo com a escolha dos pontos de cruzamento:

- Um ponto: um ponto de cruzamento é escolhido e a partir deste ponto as informações genéticas dos pais serão trocadas. As informações anteriores a este ponto em um dos pais são unidas às informações posteriores a este ponto no outro pai, como é mostrado na figura D.2.

- Multiponto: é uma generalização da forma anterior, aplicando vários pontos de corte, produzindo uma diversidade maior na população.
- Uniforme: não utiliza pontos de cruzamento, mas define um parâmetro global, indicando a probabilidade de cada variável ser trocada entre os pais.

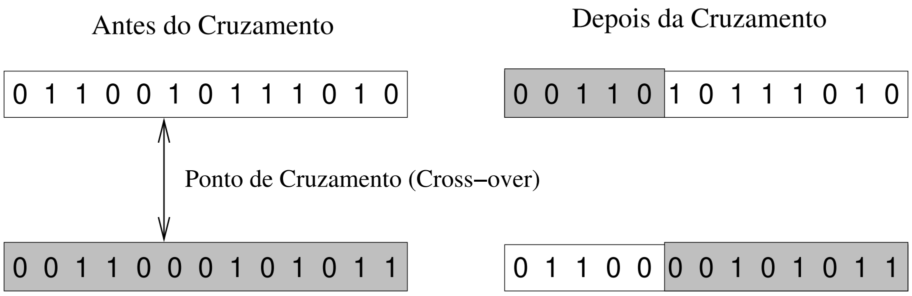


Figura D.2: Diagrama de uma operação de cruzamento.

D.5.3 Mutação

O operador de mutação é importante para garantir a diversidade genética da população, alterando aleatoriamente um ou mais componentes (genes) de uma estrutura, mostrado na figura D.3. O objetivo principal do operador mutação é a introdução de novos indivíduos à população. Assim, a mutação garante que a probabilidade de se chegar a qualquer ponto do espaço de busca é diferente de zero. Uma alteração drástica no genótipo propicia a alteração da direção da busca, permitindo a saída de pontos de máximo locais indicados pelo operador de seleção. O operador de mutação é aplicado aos indivíduos com uma probabilidade dada pela taxa de mutação P_M, que, por ser um operador genético secundário, geralmente utiliza um valor pequeno.

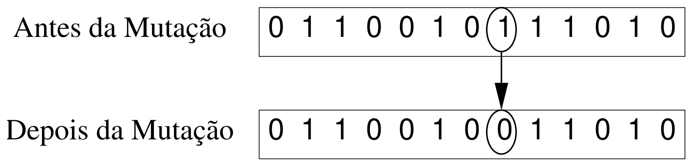


Figura D.3: Diagrama de uma operação de mutação.

D.6 Vantagens e desvantagens dos algoritmos genéticos

Podemos relacionar algumas vantagens dos algoritmos genéticos:

1. São algoritmos robustos e aplicáveis a uma grande variedade de problemas.
2. Descontinuidades ou complexidades presentes na função acarretam pouco ou nenhum efeito no desempenho da busca.
3. Não requerem conhecimento dos gradientes da função objetivo.
4. Possuem heurística eficiente para busca em universos grandes.

Como desvantagens podemos relacionar:

1. Requerem um grande número de avaliações de funções de aptidão, que pode ser computacionalmente pesado.
2. A localização de ótimo global pode ser demorado.
3. A grande quantidade de configurações possíveis pode afetar o desempenho e até não permitir o encontro da solução.