UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

CAIXA POSTAL 676
FONE/FAX: (016) 3351-8233
13565-905 – SÃO CARLOS – SP
BRASIL

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

“ Uma arquitetura de Gateway com QoS para gerenciamento de tráfego de rede considerando restrições temporais das aplicações”

Juliano Marcello

Exame de defesa apresentado ao Programa de Pós-Graduação em Ciência da Computação - PPGCC/UFSCar, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Luis Carlos Trevelin
Co-Orientador: Prof. Dr. Célio Estevan Moron

São Carlos
Julho/2009

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

M314ua

Marcello, Juliano.
Uma arquitetura de Gateway com QoS para gerenciamento de tráfego de rede considerando restrições temporais das aplicações / Juliano Marcello. -- São Carlos : UFSCar, 2009.
106 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2009.

1. Sistemas operacionais distribuídos (Computadores). 2. Qualidade de serviço. 3. Tempo real. 4. Previsão. 5. Sistemas multimídia. I. Título.

CDD: 005.44 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Uma arquitetura Gateway com Qos para gerenciamento de tráfego de rede considerando restrições temporais de aplicações"

JULIANO MARCELLO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Luis Carlos Trevelin
(Orientador - DC/UFSCar)

Prof. Dr. Célio Estevan Moron
(Co-Orientador - DC/UFSCar)

Profa. Dra. Thienne de Melo e Silva Mesquita Johnson – (USP/Leste)

Prof. Dr. Hermes Senger
(DC/UFSCar)

São Carlos
Agosto/2009

RESUMO

Aplicações com restrições temporais (*soft* e *hard real-time*) vêm se tornando comuns em nosso cotidiano. Essas aplicações variam desde jogos *online* até aplicações de controle de tráfego aéreo. As aplicações com restrições temporais que utilizam como meio de comunicação a Internet/Intranet (*e.g. streaming* de áudio e vídeo, VoIP e mensagens RPC-RT), necessitam de um mecanismo de QoS para que seus pacotes sejam entregues antes do término de seu *deadline*. Nestes casos, priorizar o tráfego dos pacotes gerados por essas aplicações em relação aos demais pacotes é uma forma de tentar minimizar a perda dos *deadlines*. Assim este trabalho apresenta uma arquitetura de *gateway* com QoS e previsão de tráfego que visa privilegiar as requisições das aplicações que possuem restrições temporais. O *gateway* classifica o tráfego, com base no padrão IEEE 802.1q, em seis classes com prioridades distintas e utiliza o modelo matemático *Box-Jenkins*, para prever o volume de tráfego para cada classe de serviço e disponibilizar recursos para o atendimento das classes de acordo com sua prioridade.

Palavras-chave: Previsão. Qualidade de serviço. Tempo real. Multimídia.

ABSTRACT

Applications with time restrictions are becoming common in our daily lives. These applications range from games to applications for controlling air traffic. Applications with time restrictions that use the Internet / Intranet as a mean of communication (e.g. streaming audio and video, VoIP and messaging PRC-RT), require that their packets be delivered before the end of its deadline. In these cases prioritize traffic from the packets generated by these applications in relation to the other packets is a way of trying to minimize the loss of deadlines. Thus this work presents an architecture of gateway with QoS and traffic forecasting that aims to focus on requests from time-restricted applications. Gateway classifies traffic based on the IEEE 802.1q standard, in six classes with different priorities and uses the Box-Jenkins mathematical model to predict the amount of traffic for each class of service and provide resources to meet the classes according to their priority.

Keywords: *Forecast, Quality of service, Real Time, Multimedia.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURA

2G Segunda Geração

3G Terceira Geração

ACF Função de Auto-correlação

ADSL *Assimetric Digital Subscriber Line*

AR Auto-Regressivo

ARIMA *Auto Regressive Integrated Moving Averages*

ARMA Auto-Regressivo de Médias Móveis

BGP4 *Border Gateway Protocol 4*

BS *Background Server*

CCM *Client Connection Manager*

CL *Client Listener*

CLP Controladores Lógicos Programáveis

DB *Data Base*

DC *Data Collector*

DeM *Decision Manager*

DM *Data Manager*

DM *Deadline-Monotonic*

DS Decision System

DS *Deferrable Server*

EDF *Earlier-Deadline-First*

FCFS *First Come First Served*

FTP *File Transfer Protocol*

HTTP *HyperText Transfer Protocol*

IEEE *Institute of Electrical and Electronic Engineers*

IETF *Internet Engineering Task Force*

LAN *Local Area Network*

LST *Least-Slack-Time-First*

MA Médias Móveis

OSI *Open Systems Interconnection*

OSPF *Open Shortest Path First*

PACF Funções de Autocorrelações Parciais

PCP *Priority Ceiling Protocol*

PHP Protocolo Herança de Prioridade

PM *Police Manager*

PS *Polling Server*

PS *Prediction System*

PW *Porcess Workers*

PW *Process Workers*

QoS *Quality of Service*

QS *Queue System*

QSM *Queuing System Manager*

QSP *Queueing System Processor*

RFC *Request for Comments*

RIP *Routing Information Protocol*

RM *Rate-Monotonic*

RPC *Remote Procedure Call*

RPCs-RT *Remote Procedure Call Real Time*

RSVP *Resource reSerVation Protocol*

SARIMA Modelo Auto-Regressivo Integrado Médias Móveis Sazonal

SI Sistemas Integrados

SL *Server Listener*

SLA *Service Level Agreement*

SS *Statistical System*

STR Sistema de Tempo Real

TOS *Type of Service*

VLAN *Virtual Local Area Network*

VOIP *Voice over Internet Protocol*

WAN *Wide Area Network*

SUMÁRIO

1 INTRODUÇÃO

A popularização de redes de banda larga (*e.g.* ADSL - *Assimetric Digital Subscriber Line*) e das redes sem fio (*e.g.* família 802.11 e 802.15) observada nos últimos anos, contribuiu para o crescimento de aplicações em tempo real. Nas redes de banda larga observa-se grande utilização de aplicações de tempo real multimídia como, por exemplo, *streaming* de vídeo e voz, em sua maioria usada para entretenimento e sistemas de monitoramento residencial e predial *online*. Já as redes sem fio permitem que um usuário possa se locomover dentro de uma área permanecendo conectado a serviços remotos.

Além desses fatores podemos citar as tecnologias 3G que são serviços oferecidos pelas operadoras de telefonia celular e possuem uma capacidade de rede maior que as tecnologias 2G por causa de uma melhora na eficiência espectral. Dentre os serviços oferecidos, há a telefonia por voz e a transmissão de dados a longas distâncias, tudo em um ambiente móvel. Normalmente, são fornecidos serviços com taxas de 5 a 10 Megabits por segundo.

Outro fator que contribuiu para o crescimento de aplicações em tempo real foi o barateamento dos dispositivos portáteis (*e.g. Notebook*) e os móveis (*e.g. Personal Digital Assistence* (PDA) e *Smartfones*). Os dispositivos móveis e portáteis estão sendo utilizados para diversos fins, tais como: educação, entretenimento, medicina e comércio. Além disso, a computação vive um momento de transição entre a computação móvel e a computação ubíqua. Weiser (1991) introduziu a área de computação ubíqua em que vislumbrou um mundo onde computadores proverão informações e serviços quando e onde forem necessários. A visão de Weiser descreveu uma proliferação de dispositivos de diferentes tamanhos, indo desde dispositivos portáteis até grandes dispositivos compartilhados. Essa proliferação

aconteceu com os dispositivos móveis e portáteis. Uma forma de melhor aproveitar os dispositivos móveis é a execução remota de determinados procedimentos em máquinas com maior capacidade de recursos, reduzindo a carga nos dispositivos móveis.

As operadoras de TV pela internet tais como Terra TV, TV IG, TV Uol, Megamax (WIKIPEDIA, 2009), também contribuíram para o crescimento das aplicações multimídia de tempo real. Elas são provedoras de serviços, que oferecem uma diversidade de canais *online*, com programação variada.

Os controladores lógicos programáveis (CLP), muito utilizados em indústrias para automação de processos em geral, surgiram no final da década de 1960 de uma necessidade da *General Motors*, em que se consumiam dias para alterar um sistema de controle baseado em relés. Com a evolução desses sistemas tornou-se possível automatizar vários processos industriais bem como monitorar e interagir com tais processos em tempo real através da rede de computadores. Em muitos casos os processos envolvendo essas aplicações possuem *deadlines* rígidos, como monitoramento e controle de reatores em usinas nucleares, necessitando ter prioridade sobre outros tipos de aplicações na rede.

Muitas das aplicações vigentes em redes residenciais ou empresariais demandam por serviços de tempo real. O problema dos diversos serviços de tempo real em uma rede é que cada serviço possui um *deadline* diferente. Além dos serviços de tempo real encontramos aplicações que não possuem *deadline* concorrendo pelos serviços da rede. Assim, faz-se necessário a implementação de alguma técnicas de QoS (*Quality of Service*) para priorizar o encaminhamento dos pacotes com menor *deadline*.

Vários trabalhos têm sido desenvolvidos com o intuito de priorizar aplicações de tempo real, como, por exemplo, Wang et al. (2005) que propõem uma arquitetura de *proxy* do tipo *environment-aware*[1] com QoS; Lei et al. (2002) e Hwang e Tseng (2006), apresentam

[1] Isto significa que os sistemas devem ter consciência da localização e da situação onde estão inseridos, e devem tirar vantagem desta informação para se auto-configurar dinamicamente de um modo distribuído.

soluções para garantia de QoS na forma de arquiteturas de *gateway* e Nobile (2007) apresenta uma arquitetura de *proxy* com QoS para RPCs-RT com o objetivo de privilegiar chamadas remotas de procedimento com maior prioridade e menor *deadline*, sem desconsiderar as demais chamadas com característica diferentes. No capítulo 4, são abordados esses e outros trabalhos relevantes a essa pesquisa.

O congestionamento causado pelo volume de chamadas que passa pelo *gateway* constitui um gargalo no cumprimento dos *deadlines*, principalmente para aquelas requisições de maior prioridade e requisitos de tempos mais críticos, comum em ambientes onde muitos dispositivos compartilham a rede de acesso.

Neste contexto a distinção entre as características de cada classe de serviço é necessária, devendo ser privilegiadas aquelas com características temporais mais rígidas. Tal tarefa deve ser realizada pelo *gateway*, no entanto, visto que o mesmo possui recursos limitados, esta é uma tarefa difícil. Além dessa restrição de recursos, quando ocorre um grande volume de chamadas, o *gateway* deverá alocar os recursos necessários para atender a demanda, organizar e escolher quais requisições devem ser transmitidas primeiro, o que também contribui para dificultar a execução das tarefas.

Este trabalho concentra-se nas aplicações de tempo real "brandas" (*soft*) como *streaming* de áudio e vídeo, VoIP e controladores de dispositivos via rede sem desconsiderar as demais aplicações, e propõe uma arquitetura de *gateway* com QoS. O *gateway* ora proposto classifica o tráfego vigente na rede em seis classes de aplicações distintas. A classificação é feita tendo como base o padrão IEEE 802.1q. Após classificar o tráfego, o *gateway* utiliza os modelos de séries temporais *Box-Jenkins* (Box e Jenkins, 1976), para prever o volume e as características futuras do tráfego das classes de serviço que, provavelmente, passarão pelo *gateway*. O modelo de previsão utilizado no *gateway* é baseado no modelo proposto por Nobile (2007). A previsão de tráfego permite a alocação antecipada e racional dos recursos

necessários para atender a demanda prevista e a escolha de políticas que visam a adaptação dos estados do *gateway* ao ambiente de rede. Para verificar a funcionalidade da arquitetura foi implementado um protótipo, o qual foi submetido a um ambiente de rede real que permitiu verificar seu comportamento sob diferentes volumes das diversas classes de aplicações.

O texto está organizado da seguinte forma:

No capítulo 2 são abordados os sistemas de tempo real e os fundamentos de qualidade de serviço; o capítulo 3 apresenta os conceitos de séries temporais, a metodologia *Box-Jenkins* utilizada para previsão e também a teoria de filas; o capítulo 4 apresenta os trabalhos relacionados a este; o capítulo 5 descreve a arquitetura de *gateway* com QoS e previsão de tráfego ora proposta; o capítulo 6 apresenta uma implementação feita para testar a arquitetura proposta e os resultados obtidos dos testes efetuados; o capítulo 7 apresenta as conclusões deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Sistemas de Tempo Real

Aplicações com requisitos de tempo real tornam-se cada vez mais comuns em nosso cotidiano, seja em nosso carro, casa, instituições financeiras, aeroportos, celulares, etc (FARINES et al, 2000). Tais aplicações variam em complexidade e cumprimento dos requisitos temporais. As mais simples podem ser encontradas em controladores inteligentes embarcados em eletrodomésticos e automóveis, já dentre as de maior complexidade estão os sistemas de defesa militar, controle de tráfego aéreo e ferroviário e em controle de plantas de indústrias químicas e nucleares. Essas aplicações que apresentam restrições de tempo são conhecidas como Sistemas de Tempo Real (STR).

Sistemas de tempo real podem ser reativos e não reativos. Sistemas não reativos são sistemas que não interagem com o ambiente. Um exemplo de sistema de tempo real não reativo são os sistemas multimídia.

Nos sistemas de tempo real reativos, o computador interage diretamente com o ambiente, não somente por meio de uma interface homem-máquina (*e.g.*, vídeo e teclado), mas, principalmente através de sensores e atuadores (MACÊDO et al, 2004). Os sensores são dispositivos eletrônicos que detectam alguma alteração no ambiente em que estão inseridos. Existem vários tipos de sensores com funções específicas como, por exemplo, sensores de temperatura. Os atuadores são dispositivos que interferem no ambiente e, assim como os sensores, existem vários tipos de atuadores usados para diversos tipos de aplicações. Podemos citar como um exemplo de atuador os hidráulicos que têm como objetivo gerar um movimento que pode ser linear ou axial.

Podemos dizer que um STR é um sistema computacional que reage a estímulos provenientes do ambiente em prazos determinados. Cada reação do sistema deve ser atendida dentro do prazo especificado. O não cumprimento de prazo causa alguma falha temporal no sistema. O comportamento do sistema de tempo real não depende só da integridade dos resultados (correção lógica ou "*correctness*"), mas também dos valores de tempo em que são produzidos (correção temporal ou "*timeliness*") (FARINES et al, 2000). Portanto um sistema de tempo real deve atender os requisitos funcionais e temporais.

2.1.1 Previsibilidade em sistemas de Tempo Real

Podemos definir um sistema como previsível se o seu comportamento (funcional e temporal) pode ser de algum modo antecipado, antes de sua execução, independente das variações de carga e falhas. Para garantirmos os requisitos temporais precisamos levantar previamente as hipóteses de carga e falhas do sistema. A carga é o volume máximo computacional gerado pelo ambiente em um determinado intervalo de tempo. As falhas descrevem as possíveis falhas em tempo de execução. Apesar das falhas o sistema ainda deve cumprir os seus requisitos temporais.

Para assumir a previsibilidade de um sistema de tempo real é preciso conhecer *a priori* o comportamento do sistema, levando em conta a pior situação de carga do sistema juntamente com suas possíveis falhas. Quanto mais próximo da realidade estiverem esses dados, mais previsível será o sistema.

É importante conhecer a carga máxima computacional gerada pelo ambiente e as hipóteses de falhas, no pior caso, para que haja um gerenciamento adequado dos recursos disponíveis a fim de evitar que ações de tempo real sofram imprevistos devido à falta de memória, CPU e banda passante na rede de comunicação, garantindo que os recursos

necessários para cumprir os requisitos temporais e funcionais estejam disponíveis. O problema que pode acontecer agora é a ocorrência de defeitos de *hardware*. De fato, uma das poucas certezas que existe na área de computação é sobre a impossibilidade de se construir sistemas totalmente imunes a defeitos, ou seja, totalmente previsíveis. O que pode ser feito, entretanto, é minimizar a ocorrência desses defeitos através de mecanismos e técnicas diversas. Por exemplo, existem técnicas que diminuem a possibilidade de se introduzir falhas já na etapa de especificação do sistema (*e.g.*, especificação e verificação formal do *software* e *hardware* dos componentes). Outras técnicas introduzem robustez no sistema de tal forma que falhas, mesmo que ocorram em tempo de execução, possam ser toleradas sem comprometer a previsibilidade (*e.g.*, replicação de componentes do sistema). Os sistemas de segurança críticos, como controle de plantas nucleares e de vôo de aeronaves, requerem uma previsibilidade próxima de 100 por cento. Por outro lado, ações de sistemas convencionais (fora do escopo de tempo real), como, por exemplo, consultas num cadastro de uma empresa, podem ser realizadas num intervalo de tempo muito menos rígido. Sendo assim, não há a necessidade de se garantir a previsibilidade em tais sistemas, apesar de ser imprescindível que sua correção (*e.g.*, o cadastro deve sempre estar consistente) seja assegurada.

Resumindo, garantir a previsibilidade do sistema é uma tarefa complexa que envolve:

1. Especificar as funções do sistema definindo precisamente seu comportamento funcional e temporal;
2. Usar ferramentas adequadas (linguagens de programação e *middleware*) para facilitar a implementação dessas especificações e reduzir possíveis erros;
3. Prover o gerenciamento dos recursos computacionais disponíveis a fim de que o sistema cumpra sua especificação;
4. Quando necessário, prover um plano de contingência dos componentes do sistema a fim de aumentar a confiança no seu funcionamento.

2.1.2 Classificação dos Sistemas de Tempo Real

Os sistemas de tempo real são classificados quanto à segurança, como (FRAGA et al, 1995) (CRUZ; LIMA, 2006):

- *Soft Real Time Systems*: nos sistemas de tempo real brandos ou não críticos, as perdas dos prazos não acarretam conseqüências graves como, por exemplo, em sistemas de *streaming* de vídeo, em que o atraso de um quadro não afetara o entendimento do vídeo. Em outras palavras o atraso de alguns quadros é tolerável.
- *Hard Real Time Systems*: nos sistemas de tempo real duros ou críticos, as perdas dos prazos acarretam conseqüências graves como, por exemplo, um sistemas que controla o trem de pouso de uma aeronave; se o trem de pouso não for acionado antes da aterrissagem teremos uma catástrofe.

2.1.3 Tarefas de Tempo Real

Eventos que são relevantes ao sistema são mapeados de forma que eles gerem estímulos (*e.g.*, sinais eletrônicos) para serem processadas pelo sistema computacional. Por exemplo, a temperatura de um forno que está sendo controlado deve ser percebida pelo sistema computacional. Para tanto, existem interfaces (*e.g.*, sensores, conversores de sinal) entre o sistema computacional e o ambiente, responsáveis por mapear os eventos relevantes ao sistema em estímulos que ativam sua computação, geralmente estruturada em um conjunto de *tarefas*. Portanto, uma tarefa pode ser definida como uma seqüência de ações executadas pelo sistema, e acionada pela ocorrência de eventos do ambiente. Como outro exemplo, considere a temperatura de uma caldeira que passou de seu limite operacional. A interface da caldeira,

representada por um sensor de temperatura, faz chegar ao sistema computacional essa informação (evento) e, a partir daí, uma ou mais tarefas corretivas são ativadas. Cada uma dessas tarefas, por sua vez, ativa outras tarefas ou responde ao ambiente através da interface (*e.g.,* atuadores, válvulas) para que haja a devida resposta ao aumento de temperatura (MACÊDO et al, 2004).

Todas as tarefas de tempo real normalmente estão sujeitas a prazos conhecidos como *deadline*. Inicialmente uma tarefa deve ser executada dentro de seu *deadline*. As tarefas podem ser classificadas quanto ao cumprimento de seus *deadlines*, como:

- Tarefas Críticas: uma tarefa é dita crítica quando o não cumprimento de seu *deadline* provocar falha catastrófica ao sistema de tempo real e/ou em seu ambiente.
- Tarefas não Críticas: essas tarefas, quando completam sua execução após seu *deadline*, no máximo implicam em uma diminuição no desempenho do sistema. As falhas temporais nesses casos não causam conseqüências muito significativas ao sistema.

Outra característica das tarefas está relacionada com a freqüência com que essas tarefas são executadas. Baseado nessas características, as tarefas são conhecidas como:

- Tarefas Periódicas: quando a seqüência de ativações dessas tarefas segue uma freqüência, uma ativação por intervalo de tempo regular. Esse intervalo de tempo é chamado de Período.
- Tarefas Esporádicas: para essas tarefas, geralmente o período é o mínimo de tempo entre duas ativações consecutivas.
- Tarefas Aperiódicas: quando a ativação dessas tarefas é feita aleatoriamente através de eventos internos ou externos.

Cada tarefa possui atributos que são parâmetros usados pelo algoritmo de escalonamento nos sistemas de tempo real. Os atributos de maior relevância são:

- Tempo de liberação: instante de tempo no qual a instância da tarefa é incluída na fila de prontos para executar.
- Tempo de chegada: é o instante de tempo em que o escalonador toma conhecimento de uma ativação da instância da tarefa.
- Tempo de início: corresponde ao instante de tempo em que uma tarefa começa seu processamento após a ativação.
- Tempo de término: é o instante em que uma tarefa termina sua execução.
- Tempo máximo de execução: tempo máximo que a instância de uma tarefa leva para executar por completo.
- *Deadline* ou *Deadline Absoluto*: instante de tempo em que a execução de uma instância de uma tarefa deve estar completa.
- *Deadline Relativo*: tempo máximo de resposta disponível a partir do tempo de lançamento. A soma do tempo de lançamento com o *deadline* relativo é igual ao *deadline* absoluto.
- *Período*: freqüência com que uma nova instância de tarefa será lançada.
- *Jitter*: em geral é assumido que tão logo uma tarefa chegue, ela é inserida na fila de prontos, porém em alguns casos, isso não acontece. O *jitter* é variação entre o tempo de chegada e o tempo de liberação da uma tarefa.

Supondo uma tarefa periódica Ti descrita pelos atributos (Ci, Di, Ji, Pi) onde Ci é o tempo máximo de execução da tarefa, Pi é o período da tarefa, Di o *deadline* da tarefa e Ji é o *jitter* da tarefa, que de certa forma representa a pior situação de liberação da tarefa, neste caso, Ji e Di são grandezas relativas ao período de cada tarefa.

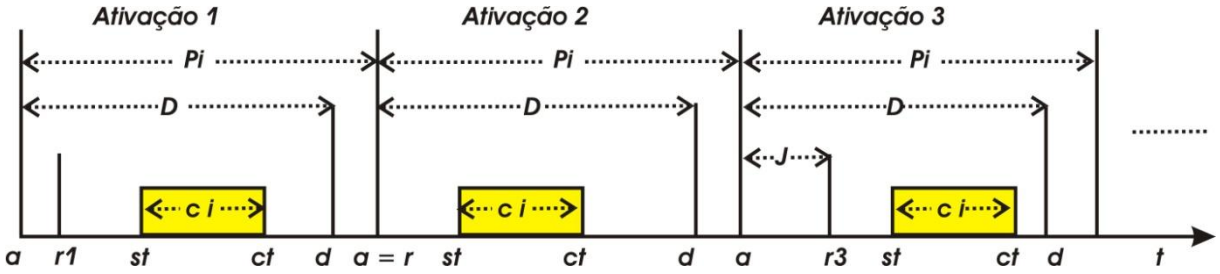


Figura 2.1: Ativações de uma tarefa periódica.

A Figura 2.1 ilustra a ativação de uma tarefa periódica em relação aos seus tempos absolutos onde: (*ai*) representa os tempos de chegada, (*ri*) os tempos de liberação, (*sti*) os tempos de inicio, (*cti*) os tempos de termino e (*di*) os *deadline* absolutos.

Uma tarefa esporádica é descrita pelos atributos (Ci, Di, Min) onde Ci é o tempo de computação, Di é o *deadline* relativo e Min corresponde ao intervalo mínimo de requisições entre duas tarefas. A Figura 2.2 ilustra a ativação de tarefas esporádicas.

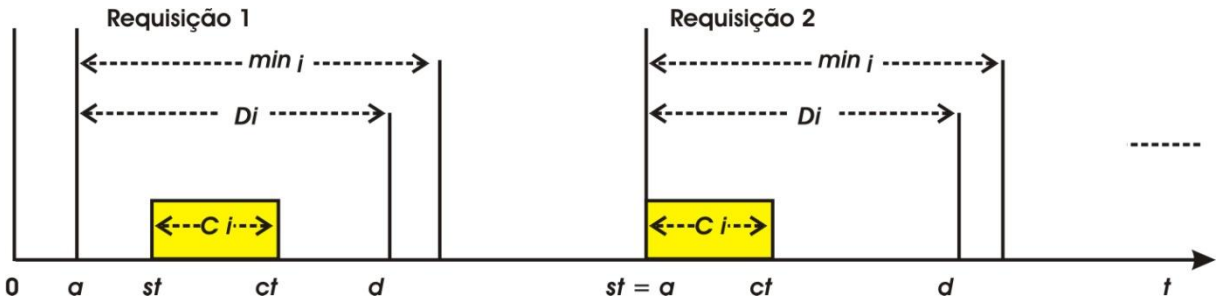


Figura 2.2: Ativação de tarefas esporádicas.

2.1.4 Escalonamento de Tempo Real

Escalonamento é o termo utilizado para descrever o procedimento de organizar as tarefas na fila de pronto definindo assim a ordem de ocupação do processador pelas tarefas da fila. O escalonador é o componente do sistema responsável pela gestão do processador em

tempo de execução. É o escalonador que implementa uma política de escalonamento definindo a ordem em que as tarefas da fila irão ocupar o processador.

Políticas de escalonamento são regras ou critérios para a ordenação das tarefas de tempo real. O escalonador utiliza essas políticas de escalonamento para produzir escalas razoáveis, que garantem o cumprimento das restrições temporais impostas por sistemas de tempo real.

Os algoritmos de escalonamento podem ser: preemptivo quando em qualquer momento a execução de uma tarefa é interrompida para que outra tarefa de maior prioridade possa ser executada ou não preemptivo quando as tarefas não são interrompidas até o termino de sua execução. Um algoritmo de escalonamento é dito estático quando o cálculo da escala é feito tomando como base os parâmetros definidos em tempo de projeto (parâmetros fixos), e dinâmicos quando o cálculo da escala é baseado em parâmetros que mudam em tempo de execução. Se a escala é produzida em tempo de execução o algoritmo é dito *on-line* caso contrário é dito *off-line*.

Uma abordagem comum entre os algoritmos de tempo real é priorizar as tarefas com maior urgência na execução. A seguir será apresenta uma comparação entre dois algoritmos de escalonamento de prioridade fixa.

Rate-Monotonic

O escalonamento *Rate-Monotonic* (RM) é um algoritmo de prioridade fixa que através de escalonadores preemptivos produz sua escala em tempo de execução. Esse algoritmo utiliza uma simples heurística de atribuição de prioridades, em que a prioridade é inversamente proporcional ao período das tarefas, ou seja, quanto maior o período da tarefa, menor é sua prioridade.

Deadline-Monotonic

O *Deadline-Monotonic* (DM), assim como o RM, também é um algoritmo preemptivo de prioridade fixa, porém ele atribui às prioridades das tarefas em ordem inversa ao *deadline* relativo, ou seja, quanto maior o *deadline* da tarefa menor sua prioridade.

Supondo duas tarefas periódicas T1 e T2 com as seguintes características descritas na tabela abaixo:

Tabela 2.1: Atributo das tarefas T1 e T2.

Tarefa Periódica	Tempo de Computação Ci	Período Pi	Deadline Di
T1	30ms	80ms	70ms
T2	30ms	120ms	50ms

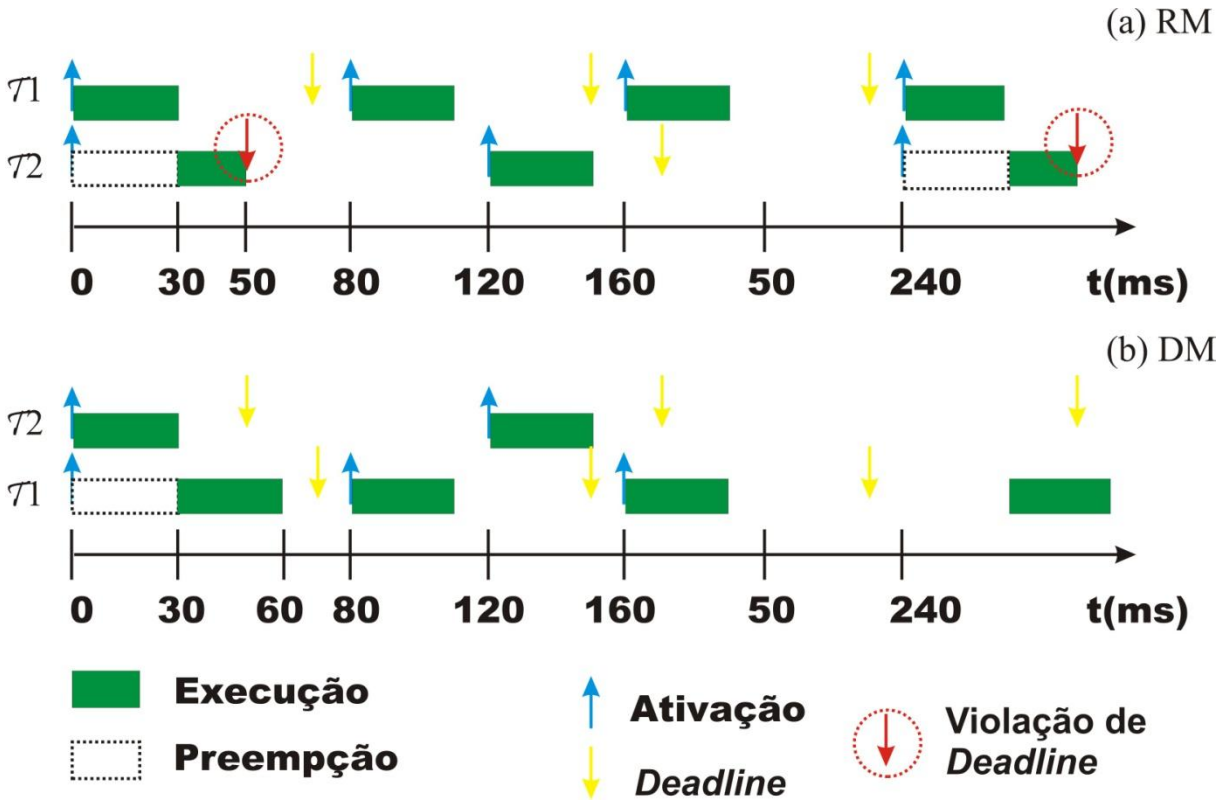


Figura 2.3: Ilustração de escalonamentos: (a) RM e (b) DM.

A Figura 2.3 ilustra o escalonamento das tarefas periódicas T1 e T2 através dos algoritmos RM e DM. Seguindo os critérios do RM a tarefa T1 é mais prioritária, já sob os critérios do DM a tarefa T2 é a mais prioritária. Como podemos observar, esse conjunto de tarefas é escalonável pelo DM, mas não pelo RM. De fato, a tarefa T2 viola seu *deadline* na primeira e terceira ativações, momento em que T1 é mais prioritária. Esse exemplo ilustra o fato de que escalonadores DM possuem melhor desempenho que o RM (LIU, 2000) (BENGTSSON et al, 1996).

A seguir será apresentado um comparativo entre dois algoritmos de prioridade dinâmica.

Earlier-Deadline-First

O *Earlier-Deadline-First* (EDF) é um algoritmo de escalonamento de prioridade dinâmica e escala produzida através de um escalonador preemptivo em tempo de execução. A política de escalonamento EDF define a tarefa de maior prioridade a tarefa que possui *deadline* mais próximo do tempo atual. A fila de prontos é reordenada a cada chegada de tarefas na fila considerando a nova distribuição de prioridades.

Least-Slack-Time-First

O algoritmo *Least-Slack-Time-First* (LST) atribui prioridades dinamicamente e em tempo de execução como o EDF e também utiliza um escalonador preemptivo. A política de escalonamento do LST ordena as prioridades das tarefas com base nos valores das respectivas folgas. Essa folga de tempo (*slack*) é definida como $d_i - t - c_i(t)$, onde d_i é o *deadline* absoluto da tarefa Ti, ou seja, o tempo de término máximo para a tarefa ser executada com

sucesso; e $c_i(t)$ é o que resta para ser executado da tarefa Ti a partir do instante t. Tal folga de tempo é monitorada em tempo de execução para que as prioridades relativas entre as tarefas em execução sejam ajustadas. Devido ao alto custo computacional que tal abordagem iria impor, implementações de LST monitoram o tempo de folga apenas quando tarefas são ativadas ou têm suas execuções completadas.

Tabela 2.2: Atributos das Tarefas periódicas.

Tarefa Periódica	Tempo de Computação Ci	Período Pi	Deadline Di
T1	30ms	10ms	20ms
T2	80ms	45ms	79ms

A Figura 2.4 ilustra o comportamento das tarefas de acordo com os atributos da tabela 2 comparando os algoritmos EDF e LST. Ambas as tarefas são ativadas no instante 0. Nesse momento a prioridade de T1 é maior que a de T2 para ambos os escalonadores, pois d1 = 20ms é menor que d2 = 79ms (critério do EDF) e 20 - 0 - 10 = 10ms é menor que 79 – 0 - 45 = 34ms (critério do LST). Quando T1 termina sua execução, T2 inicia a sua até sofrer preempção, aos 30ms, pois, na sua segunda ativação, T1, mais uma vez, tem maior prioridade para ambos os escalonadores. Apenas na sua terceira ativação, aos 60ms, é que as decisões de escalonamento do EDF e do LST diferem. O EDF atribui maior prioridade a T2, pois nesse instante d2 = 79ms é menor que d1 = 80ms. Para o LST, no entanto, T1 é mais prioritária, pois seu tempo de folga (80 - 60 - 10 = 10ms) é menor que o de T2 (79 - 60 - 5 = 14ms).

O LST é considerado ótimo em relação aos critérios de prioridades dinâmica porém possui *overhead* maior que EDF.

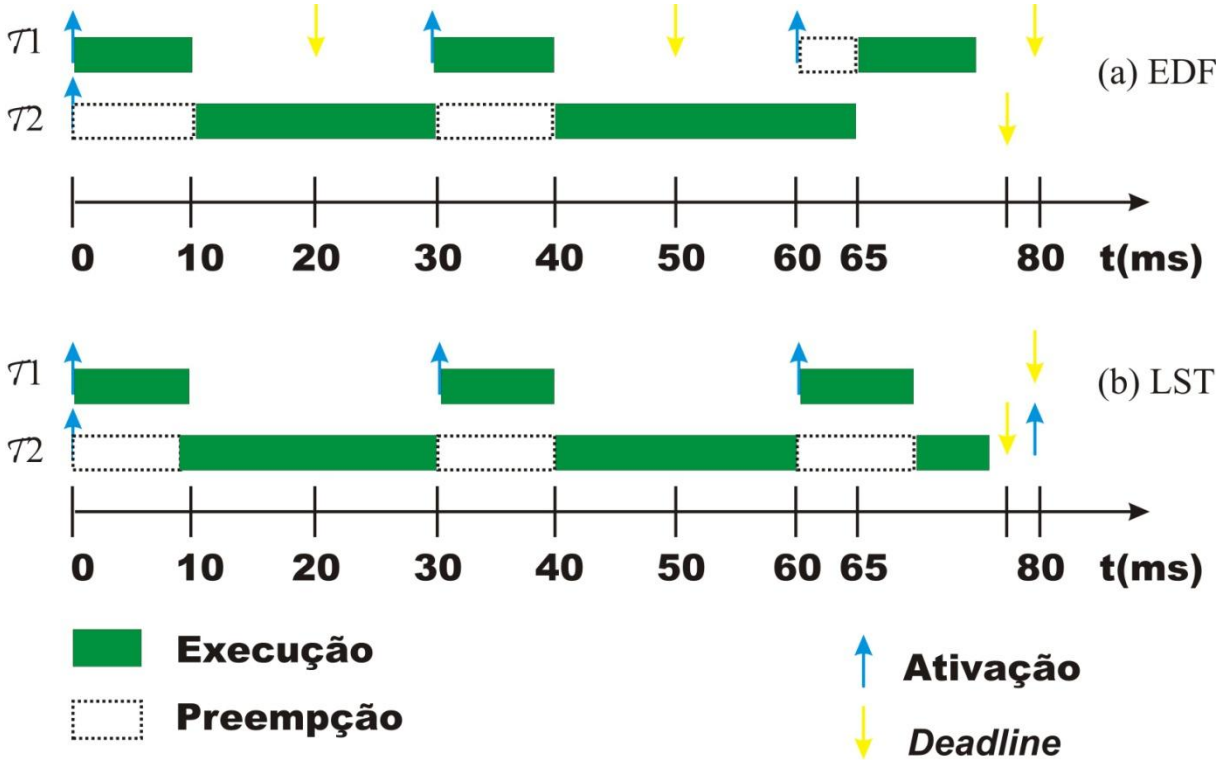


Figura 2.4: Ilustração de escalonamentos: (a) EDF e (b) LST.

Teste de Escalonabilidade

O teste de escalonabilidade é importante para definir se um conjunto de tarefas de tempo real é escalonável, ou seja, se existe para esse conjunto de tarefas uma escala razoável. Esse teste se propõe a verificar se determinado sistema tem todos os seus *deadlines* críticos satisfeitos quando escalonado de acordo com determinado algoritmo. Geralmente este teste é feito em tempo de projeto e baseia-se no conhecimento sobre as tarefas periódicas e esporádicas do sistema. Tais testes são dados por meio de equações que se satisfeitas podem informar se determinado sistema é escalonável. Seja qual for o teste de escalonabilidade usado, ele dará apenas um tipo de informação: se há ou não chances de existir violação de *deadline*. Esta informação é útil, mas pode não ser suficiente (CRUZ; LIMA, 2006).

Duas abordagens são geralmente usadas para se analisar a escalonabilidade de sistemas de tempo real. Uma se baseia no cálculo da utilização máxima do processador e a outra verifica se os tempos de resposta das tarefas no pior caso ultrapassam seus *deadlines*.

Para o cálculo de utilização máxima do processador, suponha um conjunto de n tarefas periódicas e preemptíveis T = {T1,T2,T3,..,Tn}, que não compartilham recursos e são escalonadas num sistema com único processador. Cada tarefa Ti é ativada no sistema a uma taxa de 1/T_i e tempo de execução máximo C_i. Em outras palavras, cada tarefa *utiliza* no máximo $u_i = C_i/T_i$ do potencial do processador. Assim, o fator de utilização máximo desse conjunto de tarefas é dado na forma geral pela equação:

$$U = \sum_{\forall Ti \in T} \frac{Ci}{Ti} \tag{1.1}$$

Para a abordagem que verifica os tempos de resposta, vamos assumir um modelo de tarefas que contempla apenas tarefas periódicas, que não compartilham recursos, que podem sofrer preempção e que possuem *deadlines* menores ou iguais aos seus respectivos períodos. Essas hipóteses simplificadoras do modelo serão removidas ao longo da descrição.

Intuitivamente, pode-se verificar que o pior tempo de resposta de cada tarefa $T_i \in T$ ocorre quando é necessário C_i unidades de tempo para executar T_i e que T_i sofre interferência máxima (através de preempção) das tarefas que possuem prioridades mais elevadas que T_i. Seja A_i essa interferência. Então, o tempo máximo de resposta (R_i) de T_i é dado por:

$$Ri = Ci + Ai \tag{1.2}$$

A interferência Ai ocorre quando todas as tarefas Tj ∈ T, que têm prioridades maiores que Ti, são ativadas no sistema no mesmo instante que Ti. Nesse cenário, note que cada Tj pode ocorrer $\left\lceil \frac{Ri}{Tj} \right\rceil$ vezes durante o tempo de resposta de Ti, onde a operação $\left\lceil \frac{Ri}{Tj} \right\rceil$ representa o menor inteiro que é maior igual ou igual a $\frac{Ri}{Tj}$. No pior caso, para cada uma dessas vezes, a execução de T_j interfere C_j na execução de T_i. Assim, o termo A_i é dado por:

$$Ai = \sum_{j \in hp(i)} \left\lceil \frac{Ri}{Tj} \right\rceil .Cj \tag{1.3}$$

Em que hp(i) é o conjunto de tarefas mais prioritárias que Ti. Pelas equações (2) e (3), têm-se que (AUDSLEY et al, 1993):

$$Ri = Ci + \sum_{j \in hp(i)} \left\lceil \frac{Ri}{Tj} \right\rceil .Cj \tag{1.4}$$

Note que o termo Ri aparece em ambos os lados da equação 1.4. Para resolvê-la aplica-se a relação de recorrência dada pela equação 1.5 num procedimento iterativo (AUDSLEY et al, 1993). A iteração pode se iniciar com $r_i^0 = Ci$, onde r_i^0 é a k-ésima aproximação de Ri. O procedimento se encerra quando $r_i^{k+1} > D_i$ ou se $r_i^{k+i} = r_i^k$ para algum k. No primeiro caso, Ti não é escalonável, enquanto que o segundo significa que $R_i = r_i^k$.

$$r_i^{n+1} = Ci + \sum_{j \in hp(i)} \left\lceil \frac{r_i^n}{Tj} \right\rceil .Cj \tag{1.5}$$

Para exemplificar o uso deste teste de escalonabilidade, considere o conjunto de tarefas periódicas T1, T2, T3 com seus atributos descritos na tabela 2.3 e escala produzida segundo política do algoritmo DM, o qual foi escolhido por ser de prioridade estática.

Tabela 2.3: Tarefas periódicas e seus atributos.

Tarefa Periódica	Tempo de Computação Ci	Período Pi	Deadline Di
T1	2ms	10ms	6ms
T2	2ms	10ms	8ms
T3	8ms	20ms	16ms

Observa-se que a prioridade de T1 > T2 > T3. Os tempos de resposta são determinados a partir da aplicação da equação 1.4. A tarefa T1, por ser mais prioritária não sofre interferência das demais tarefas e seu tempo de resposta é igual a $C_{T1} = 2$. Portanto T1 é escalonável, pois seu tempo de resposta máximo é menor que seu *deadline* relativo ($D_{T1} = 6$). A tarefa T2 sofre interferência de T1, assim o cálculo de seu tempo de resposta é obtido aplicando a equação 1.4:

$$R_{T2}^{0} = C_{T2} = 2ms$$

$$R_{T2}^{1} = 2 + \left\lceil \frac{2}{10} \right\rceil .2 = 4ms$$

$$R_{T2}^{2} = 2 + \left\lceil \frac{4}{10} \right\rceil .2 = 4ms$$

A tarefa T2 que apresenta $R_{T2} = 4$ também é escalonável ($R_{T2} \leq D_{T2}$). A tarefa T3 sofre interferência de T1 e T2, o cálculo de R_{T3} é:

$$R_{T3}^{0} = C_{T3} = 8ms$$

$$R_{T3}^{1} = 8 + \left\lceil \frac{8}{10} \right\rceil .2 + \left\lceil \frac{8}{10} \right\rceil .2 = 12ms$$

$$R_{T3}^{2} = 8 + \left\lceil \frac{12}{10} \right\rceil .2 + \left\lceil \frac{12}{10} \right\rceil .2 = 16ms$$

$$R_{T3}^{3} = 8 + \left\lceil \frac{16}{10} \right\rceil .2 + \left\lceil \frac{16}{10} \right\rceil .2 = 16ms$$

A tarefa T3 também é escalonável apresentando um tempo de resposta igual ao seu *deadline* relativo.

Nos modelos apresentados até aqui assumimos que as tarefas periódicas são liberadas sempre no início de cada período. Contudo isso nem sempre corresponde à realidade. Algumas tarefas podem sofrer atrasos em suas liberações. Esses atrasos podem ser expressos no pior caso como *release jitter* (J).

Para que se possa considerar o *jitter* no cálculo do tempo de resposta devemos fazer as devidas modificações na equação 1.4. Para isso deve-se entender o conceito de período ocupado (*busy period*). Um período ocupado i corresponde a uma janela Wi onde ocorre a execução contínua de tarefas com maior prioridade ou igual a i. O período i é associado a uma tarefa Ti, tendo seu início na liberação da instância de Ti, pressupondo que todas as tarefas com prioridade maior que i estejam na fila de pronto.

Considerando Wi o limite máximo das ocorrências de Tj nesse intervalo, dado por $\left\lceil \frac{W_i}{T_j} \right\rceil$. Se uma instância de Tj anterior ao início de Wi tem um *jitter* Ji na sua liberação, e a interferência sofrida pela tarefa Ti está associada a Wi, o número de ativações de Tj que interferem em Ti passa a ser $\left\lceil \frac{W_i + J_j}{T_j} \right\rceil$. Assim o cálculo de Wi passa a ser (FARINES et al, 2000):

$$W_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i + J_j}{T_j} \right\rceil . C_j \qquad (1.6)$$

Assim Wi é o tempo de resposta máximo de Ti sem considerar Ji. Se Ti também sofrer *jitter*, deve-se soma-lo ao tempo de resposta. Deste modo o tempo de resposta final Ri é dado por:

$$R_i = W_i + J_i \tag{1.7}$$

2.1.5 Compartilhamento de recursos entre tarefas

Supomos até agora que as tarefas não compartilham recursos, o que restringe muito a elaboração de um sistema concorrente. Em sistemas baseados em prioridade que compartilham recursos há a possibilidade de tarefas mais prioritárias ficarem esperando (bloqueadas) por recursos que estão sendo usados por tarefas menos prioritárias. Esse problema é chamado de inversão de prioridade.

Inversão de Prioridade

Para exemplificar, vamos supor um conjunto de tarefas T = {T1, T2, T3} em que a prioridade de T1 é maior que T2 e a prioridade de T2 maior que T3. Vamos imaginar que as tarefas T1 e T3 compartilham um recurso, e que a tarefa T3 começa a executar e aloca o recurso. Em seguida a tarefa T1 inicia sua execução e fica bloqueada no instante em que tenta acessar o recurso. Instantes depois T2 inicia sua execução causando a preempção de T3. A partir daí T1 sofre interferência não apenas de T3 com quem compartilha recurso, mas também de T2 que não compartilha recurso e possui prioridade menor que T1. Note que quaisquer tarefas que tenham prioridades intermediárias (entre T_1 e T_3) poderão interferir na

execução da tarefa mais prioritária. Conseqüentemente, o comportamento temporal de T_1 pode se tornar imprevisível (SPRUNT et al, 1989).

A inversão de prioridade não pode ser completamente evitada, mas existem técnicas que amenizam seus efeitos. Uma solução simples é impedir a preempção de tarefas que estejam executando em uma região crítica. Assim T3 não sofreria preempção de T2 e T1 esperaria somente o término de T3. Esse protocolo resolve o problema, mas não é recomendado, pois a execução de T3 interfere em todas as tarefas do sistema independente do compartilhamento de recursos. Duas soluções eficazes para resolver o problema de inversão de prioridade são: Protocolo Herança de Prioridade e *Priority Ceiling Protocol* (SHA et al, 1990).

Protocolo Herança de Prioridade

O Protocolo Herança de Prioridade (PHP) determina que as prioridades das tarefas sejam atribuídas por alguma política de prioridade fixa (RM, DM, etc.) e prioridade dinâmica, somente quando existem ações de bloqueio no sistema. Se não ocorrer bloqueio no sistema as tarefas são escalonadas de acordo com suas prioridades definidas em tempo de projeto. Se uma dada tarefa T1 é bloqueada em um semáforo por uma tarefa T2, T2 herdará a prioridade de T1 até o término de sua execução na seção crítica (SHA et al, 1990).

O PHP pode causar as seguintes situações de bloqueio para tarefas mais prioritárias:

- Bloqueio direto: ocorre quando uma tarefa mais prioritária tenta acessar um recurso compartilhado já bloqueado por uma tarefa menos prioritária.
- Bloqueio por herança: quando a execução de uma tarefa de prioridade intermediária é bloqueada por uma tarefa que herdou a prioridade de uma tarefa mais prioritária.

- Bloqueio transitivo: este ocorre somente quando existem seções críticas aninhadas. Supondo um conjunto de tarefas T ={T1, T2, T3, T4 } em que, T1 é a de maior prioridade seguida por T2 e T3 tem prioridade menor que T2 e maior que T4. T2 e T3 possuem seções aninhadas. T1 é bloqueada por T2 que por sua vez é bloqueada por T3 e finalmente T3 é bloqueada por T4. T1 só retorna sua execução quando houver a liberação de T4, T3, T2 respectivamente. Esse tipo de bloqueio pode levar a situações de *deadlocks*.

Priority Ceiling Protocol

O *Priority Ceiling Protocol* (PCP) é uma extensão do PHP em que é adicionada uma regra de controle sobre os pedidos de entrada em exclusão mútua, a fim de limitar o número de inversão de prioridade e evitar a formação de cadeias de bloqueio e *deadlocks*. Esse protocolo assume que tarefas são escalonadas baseadas em prioridades fixas, assim como o DM, e que os recursos por elas requisitados são conhecidos (SHA et al, 1990).

Basicamente o PCP garante uma inversão de prioridade por ativação causada por tarefas menos prioritárias. Assim conflitos de acesso a recursos compartilhados são resolvidos sem a necessidade do uso de trancas (*locks*) explícitas. A simples manipulação de prioridades funciona como trancas implícitas.

Todos os recursos acessados em exclusão mútua possuem um valor de prioridade teto (*Ceiling*) que é o valor da tarefa mais prioritária que acessa o recurso. Deste modo, a tarefa só entra na seção crítica se sua prioridade for maior que a prioridade teto de qualquer recurso previamente bloqueado, salvo recursos bloqueados pela tarefa requerente. Quando todos os recursos estiverem livres de bloqueio o primeiro acesso sempre é permitido.

Esse protocolo introduz uma outra forma de bloqueio além das introduzidas no PHP, conhecida como bloqueio *ceiling* em que uma tarefa fica bloqueada por não possuir prioridade superior à prioridade teto entre os recursos ocupados.

2.1.6 Escalonamento de tarefas Aperiódicas

Até agora apresentamos técnicas de escalonamento referentes a modelos de tarefas periódicas. As tarefas aperiódicas que apresentam um intervalo mínimo entre ativações são classificadas como esporádicas (SPRUNT et al, 1989). Essas tarefas possuem um comportamento temporal determinista, o que facilita a obtenção de garantias em tempo de execução. As tarefas aperiódicas que não possuem tempo de chegada conhecido e sem um intervalo mínimo entre ativações, não possuem um comportamento temporal determinístico.

Abordaremos as técnicas de servidores de prioridade fixas para escalonamento de tarefas aperiódicas baseadas no RM. O RM foi escolhido por sua heurística de atribuição de prioridade ser baseada no período de ativação das tarefas. Essa característica do protocolo permite a inserção de uma tarefa auxiliar para tratar as tarefas aperiódicas, como veremos a seguir.

Background Server

Esta técnica consiste em atender as tarefas aperiódicas quando a fila de pronto envolvendo tarefas periódicas está vazia. As prioridades são atribuídas de acordo com as políticas do RM. São definidas prioridades mais altas para tarefas periódicas e prioridades menores para tarefas aperiódicas. O *Background Server* (BS) apresenta tempos de resposta para tarefas aperiódicas muito altos, devido a sua política de atribuição de prioridade. Quanto

maior a carga envolvendo tarefas aperiódicas, menor é a utilização disponível para o servidor (LEHOCZKY et al, 1987).

O BS é uma técnica simples de ser implementada, porém só é aplicável quando as tarefas aperiódicas não são críticas e a carga referente a tarefas periódicas não é alta.

Polling Server

O *Polling Server* (PS) define uma tarefa periódica para atender a carga das tarefas aperiódicas (SPRUNT et al, 1989). Esta tarefa é chamada de Servidora *Polling*, e possui um período Ps e tempo de computação Cs. Como as demais tarefas, essa tarefa tem sua prioridade definida de acordo com as políticas definidas no RM. Em cada ativação, a tarefa servidora executa as tarefas pendentes aperiódicas dentro do limite Cs. Se não houver nenhuma tarefa aperiódica pendente a tarefa servidora é suspensa até sua próxima ativação. Neste caso o tempo de computação Cs é entregue para a execução de tarefas periódicas pendentes. Caso chegue uma tarefa aperiódica logo após a suspensão da tarefa servidora, esta deve aguardar a próxima ativação da tarefa servidora.

Deferrable Server

O *Deferrable Server* (DS) também se baseia na criação de uma tarefa periódica para atender a carga gerada pelas tarefas aperiódicas. Assim como PS, o DS atribui suas prioridades de acordo com o RM. O que difere o DS do PS é a conservação do tempo destinado para o processamento das tarefas aperiódicas, mesmo quando não existir requisições durante a ativação do DS. Deste modo, tarefas aperiódicas que chegarem durante o período de DS poderão ser atendidas enquanto Cs não esgotar o período correspondente.

Devido a essa característica, o DS fornece tempos de resposta melhores para tarefas aperiódicas que o PS.

Como visto nesta seção diversas técnicas podem ser aplicadas a fim de garantir os requisitos temporais das tarefas. Neste trabalho pretendemos aplicar os conceitos vistos anteriormente para priorizar o tráfego de aplicações com requisitos temporais tais como multimídia, VoIP e RPC-RT. Os pacotes gerados por estas aplicações possuem requisitos temporais quanto à entrega. De forma semelhante às tarefas, os pacotes devem ser encaminhados de acordo com os requisitos temporais, ou seja, devem ter prioridade em relação aos pacotes sem requisitos temporais.

2.2 Qualidade de Serviço

Em redes de computadores uma seqüência de pacotes da origem até o destino é chamada de fluxo. Tradicionalmente os pacotes são encaminhados no esquema melhor esforço, ou seja, cada usuário da rede envia seus dados e compartilha a largura de banda com todos os fluxos de dados dos outros usuários. O fluxo realiza a melhor forma possível para chegar ao seu destino, conforme as rotas definidas e a largura de banda que estiver disponível. Entretanto, quando a demanda por serviços é muito grande e a capacidades dos equipamentos não é suficiente para atender imediatamente as requisições dos clientes é preciso determinar parâmetros para atender a necessidade de cada fluxo. Tais necessidades são caracterizadas por quatro parâmetros principais descritos abaixo (TANENBAUM, 2003):

- Confiabilidade: nenhum *bit* pode ser entregue de forma incorreta. Em geral, esse objetivo é alcançado calculando-se o total de verificação de cada pacote e conferindo-se o total de verificação no destino. Se um pacote for danificado em trânsito, ele não será confirmado e será retransmitido mais tarde. Aplicações como transferência de

arquivos e *login* remotos são exemplo de aplicações que tem rigidez quanto confiabilidade.

- Retardo: é o atraso dos pacotes. Aplicações de tempo real, como telefonia e videoconferência, têm requisitos rígidos quanto ao retardo.
- Flutuação: consiste na chegada de pacotes com intervalos de tempo irregulares entre eles. O vídeo e o áudio são extremamente sensíveis à flutuação. Se um usuário estiver assistindo a um vídeo transmitido pela rede e os quadros estiverem todos atrasados exatamente 3 segundos, não haverá nenhum dano. Porém, se o tempo de transmissão variar ao acaso entre 1 e 3 segundos, o resultado será terrível. No caso do áudio, até mesmo uma flutuação de até alguns milissegundos será bastante audível.
- Largura de banda: as aplicações diferem quanto suas necessidades de largura de banda. O correio eletrônico e o *login* remoto não necessitam de muita largura de banda, mas todas as formas de vídeo exigem um grande volume desse recurso.

Segundo Melo (2001), qualidade de serviço é entendida como a capacidade da rede, através dos mecanismos de reserva de largura de banda e priorização de tráfego, em fornecer garantias de que determinados fluxos de tráfego irão ter tratamento diferenciado.

Existem várias técnicas que auxiliam na obtenção de qualidade de serviço em redes de computadores. Abordaremos as com maior relevância para este trabalho.

2.2.1 Superdimensionamento

Uma solução prática é super-dimensionar os recursos, assim com recursos sobrando os equipamentos da rede podem atender as solicitações de serviços imediatamente com confiabilidade, sem retardo, sem flutuação e com largura de banda suficiente para atender ao trafego esperado. Porém, o problema com essa solução é seu custo (TANENBAUM, 2003).

2.2.2 Armazenamento em buffers

O armazenamento em *buffers* consiste em armazenar os fluxos em *buffers* antes da entrega. Essa solução não afeta a confiabilidade ou a largura de banda e aumenta o retardo, mas reduz a flutuação. Essa técnica pode aumentar ligeiramente o atraso na reprodução de áudio ou vídeo, mas evita criar um incômodo intervalo no áudio ou vídeo (TANENBAUM, 2003).

2.2.3 Serviços integrados

A IETF (*Internet Engineering Task Force*) dedicou um grande esforço à criação de uma arquitetura para fluxo de aplicações multimídia. Esse trabalho resultou em mais de duas dezenas de RFCs (*Request for Comments*), começando com as RFCs 2205 a 2210 (TANENBAUM, 2003). O principal protocolo da IETF para a arquitetura de serviços integrados é o RSVP (*Resource reSerVation Protocol*), descrito na RFC 2205 e em outras.

Segundo Nobile (2007), a reserva de recursos é o passo inicial executado pela aplicação antes de transmitir o fluxo. Os recursos devem ser previamente configurados ao longo do caminho do fluxo antes da transmissão dos dados. O modelo de Sistemas Integrados (SI) otimiza a utilização da rede e dos recursos para aplicações pouco tolerantes a falhas e congestionamento. O bom funcionamento desse tipo de aplicação requer que os recursos estejam disponíveis e sejam suficientes quando forem necessários na transmissão.

Para conseguir cumprir com os requisitos das aplicações, o modelo de SI divide o tráfego de maneira racional para as aplicações tradicionais e as aplicações que necessitam de

QoS. Para suportar o modelo de serviços integrados, um roteador precisa ser capaz de propiciar a QoS apropriada para cada fluxo de dados, de acordo com o modelo do serviço.

Neste modelo os recursos devem ser reservados fim-a-fim para garantir a QoS adequada.

RSVP (*Resource reSerVation Protocol*)

Neste protocolo cada grupo recebe um endereço de grupo, assim para transmitir dados a um grupo, um transmissor coloca o endereço do grupo no pacote. A reserva de recurso deste protocolo é dada pelo receptor, uma vez que o mesmo deve iniciar a solicitação de reserva. A reserva é dada pela rota inversa a do fluxo até chegar à fonte, ou em algum nó intermediário com as mesmas necessidades. Em cada *hop* (salto), o roteador detecta a reserva e guarda a largura de banda necessária. Se não houver banda suficiente disponível é reportada falha (TANENBAUM, 2003).

No momento da reserva o receptor pode especificar mais de uma origem a partir das quais deseja receber informações. O receptor também determina se essas opções serão fixas durante o período de reserva ou não, podem assim deixá-las em aberto para alterar as origens mais tarde. Essas informações são utilizadas pelos roteadores para otimizar o planejamento da utilização da largura de banda.

2.2.4 DiffServ - Serviços Diferenciados

Uma abordagem mais simples criada pelo IETF para fornecer qualidade de serviço é conhecida como serviços diferenciados ou qualidade de serviços baseados em classes. Diferente da solução baseada em reserva de serviço essa abordagem não necessita reservar

antecipadamente os recursos. Neste caso, serviços diferenciados podem ser oferecidos por um conjunto de roteadores que formam um domínio administrativo configurados previamente para um conjunto de classes de serviços com regras de encaminhamento correspondente as necessidades de cada classe. Essas regras denominam-se SLA – *Service Level Agreement*. Um SLA determina as classes de serviços suportadas e a quantidade de tráfego na banda entre os domínios. Os domínios podem definir um SLA estático ou dinâmico, sendo que, neste último caso, um protocolo de sinalização e controle será necessário para o gerenciamento da banda.

No DiffServ o antigo campo TOS (*Type of Service)* do cabeçalho do protocolo IP passa a ser chamado DS e a carregar as características de serviço do pacote. O campo DS determina qual será o tratamento dado ao pacote de acordo com a classe de QoS, ou seja, por meio do valor estipulado neste campo é possível que o roteador saiba quais são as prioridades atribuídas ao pacote (NOBILE, 2007). A Figura 2.5 mostra o cabeçalho do protocolo IP versão 4 (quatro) com o campo DS em destaque.

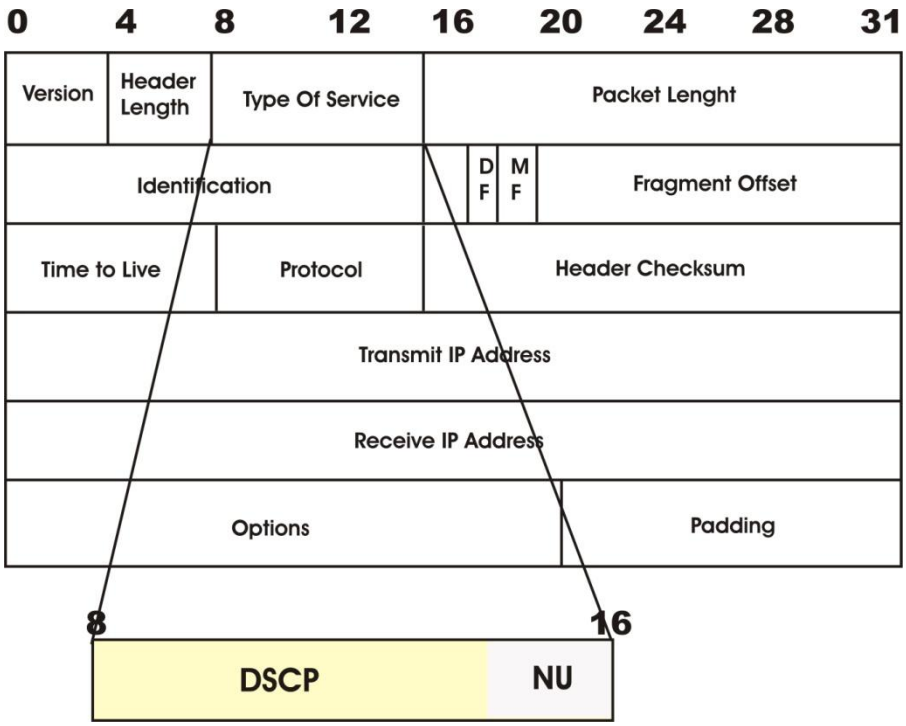


Figura 2.5: Cabeçalho IPv4 (NOBILE, 2007).

Dois novos tipos de serviços especiais surgiram juntamente com o modelo de serviços diferenciados: serviços assegurados e os serviços *premium*. Serviços assegurados são os serviços para clientes que precisam de segurança para seus provedores de serviços no momento que haja um congestionamento. E os serviços *premium* são para aplicações que necessitam de baixo *jitter* (TANENBAUM, 2003).

3 FUNDAMENTOS MATEMÁTICOS

3.1 Séries Temporais

Para Morettin e Toloi (1987), o estudo de séries temporais baseia-se na idéia de que as observações passadas da série contêm informação sobre o seu padrão de comportamento futuro. As séries temporais são compostas por quatro elementos:

1. Tendência: indica o sentido de deslocamento da série ao longo do tempo;
2. Ciclo: movimento ondulatório que ao longo do tempo tende a ser periódico;
3. Sazonalidade: movimento ondulatório de curta duração;
4. Ruído aleatório ou erro: compreende a variabilidade intrínseca aos dados e não pode ser modelado.

Segundo Morettin e Toloi (1987) todas as possíveis observações de um evento é um processo estocástico, sendo assim cada possível trajetória observada ao longo do tempo é uma série temporal. Assim pode-se dizer que uma série temporal é qualquer conjunto Z de observações ordenadas no tempo, $Z = \{Z_t, t = 1, 2,...,N\}$ sendo t o índice de tempo e N é o número de observações. Dentre as várias razões de se estudar séries temporais podemos citar:

- *Controle de processos:* estuda o mecanismo gerador da série temporal, visando ter controle sobre os processos;
- *Descrição:* compreender o comportamento da série através da geração de gráficos para representá-la, e assim verificar a existência de tendência, variação sazonal, alterações estruturais, etc;
- *Explicação:* usar a variação em uma série para explicar a variação em outra série;
- *Predição:* Prever valores futuros da série;

Neste trabalho a principal finalidade é o estudo da série temporal relativa ao volume de requisições feitas ao servidor ao longo do tempo, para que possa ser feita previsões de tráfego futuro a partir da série observada.

Existem vários modelos utilizados para descrever séries temporais. Alguns modelos se ajustam melhor a séries do que outros. O método *Box-Jenkins* (BOX e JENKINS, 1976) apresenta um conjunto de modelos que são aplicados em séries com comportamento estacionário, não estacionário ou sazonal. Esta característica da metodologia *Box-jenkins* torna este método mais interessante que os baseados em suavização exponencial.

3.2 Metodologia dos Modelos de Box-Jenkins

A metodologia de *Box-Jenkins*, descrita na década de 70, diz que a partir do uso da estrutura de correlação seriada ou autocorrelação que, geralmente, há entre os valores da série, cada valor pode ser explicado por valores anteriores. Os modelos *Box-Jenkins* são amplamente utilizados para previsão em diversas áreas tais como: financeiras, médicas, ambientais e engenharia (ABDEL-AAL e AL-GARNI, 1997).

Os modelos de *Box-Jenkins* são modelos matemáticos utilizados para captar a correlação ou autocorrelação entre os valores da série temporal e, baseados na correlação do comportamento encontrado na série realiza previsões futuras. Quando a estrutura de correlação for bem modelada, as previsões serão mais precisas. Os modelos *Box-Jenkinks* também são conhecidos como ARIMA (*Auto Regressive Integrated Moving Averages*).

3.2.1 Modelos Estacionários

Quando um processo estocástico está em equilíbrio diz se que esse modelo é estacionário. Esses processos podem ser classificados como fracamente estacionário ou fortemente estacionário. Um processo é considerado fracamente estacionário se a média e variância se mantêm constantes ao longo do tempo e a função de autocovariância depende apenas da defasagem entre os instantes de tempo. Um processo é fortemente estacionário se todos os momentos conjuntos são invariantes a translações no tempo.

Modelos Auto-Regressivo (AR)

Para o modelo auto-regressivo, a série temporal Z_t é descrita por seus valores passados regredidos e pelo ruído aleatório ε_t. Assim o modelo AR(p) é dado por:

$$\tilde{Z}_t = \emptyset_1 \tilde{Z}_{t-1} + \emptyset_2 \tilde{Z}_{t-2} + \ldots + \emptyset_p \tilde{Z}_{t-p} + \varepsilon_t \quad (3.1)$$

Em que:

$\tilde{Z}_t = Z_t - \mu$;

$\emptyset_i$ é o i-ésimo parâmetro que descreve como $\tilde{Z}_t$.

Aplicando o operador defasagem B, o modelo AR(p) dado pela equação 3.1 pode ser reescrito como $\emptyset(B) = 1 - \emptyset_1 B - \emptyset_2 B_2 - \ldots - \emptyset_p B_p$, resultando na Equação 3.2.

$$\emptyset(B)\tilde{Z}_t = \varepsilon_t \quad (3.2)$$

O modelo mais simples dessa classe de modelos é o AR(1). Sua representação algébrica é dada pela seguinte equação:

$$\tilde{Z}_t = \emptyset_1 \tilde{Z}_{t-1} + \varepsilon_t \quad (3.3)$$

Para o modelo ser estacionário é necessário que $|\emptyset_1| < 1$ e que as autocovariâncias γ_k sejam independentes (Werner e Ribeiro, 2003). Considere um modelo AR(1), então as autocovariâncias serão dadas pela equação 3.4.

$$\gamma_k = \emptyset_1^k \gamma_0 \quad (3.4)$$

Ainda considerando um modelo AR(1) as autocorrelações ρ_k são dadas por:

$$\rho_k = \frac{\gamma_k}{\gamma_0} = \emptyset_1^k \quad (3.5)$$

Para as equações 3.4 e 3.5 $k = 1,2,3,4, \ldots$

A função de autocorrelação apresenta queda exponencial para $\emptyset_1 > 0$. Para $\emptyset_1 < 0$, a função de autocorrelação também apresenta queda exponencial, mas tem seus valores variando entre sinais positivos e negativos.

Modelos de Médias Móveis (MA)

No modelo de médias móveis, a série Z_t é composta pela combinação dos ruídos brancos ε_t do período atual com aqueles ocorridos em períodos anteriores. Assim, o modelo de médias móveis de ordem q ou MA(q) é representado pela equação 3.6.

$$\tilde{Z}_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} \quad (3.6)$$

Em que:

$$\tilde{Z}_t = Z_t - \mu\,;$$

θ_i é o i-ésimo parâmetro que descreve como $\tilde{Z}_t$ se relaciona com o valor ε_{t-i}, para $i = 1,2,3,\ldots,q$.

Aplicando o operador defasagem B para o modelo MA(q) na equação 3.6 temos:

$$\tilde{Z}_t = (1 - \theta_1 B - \theta_2 B_2 - \cdots - \theta_p B_p)\varepsilon_t$$

$$\theta(B)\varepsilon_t = \tilde{Z}_t \quad (3.7)$$

A representação do modelo MA(1) é dada por:

$$\tilde{Z}_t = \varepsilon_1 + \theta_1 \varepsilon_{t-1} \quad (3.8)$$

As autocorrelações ρ_k, que nada mais são do que as autocovariâncias divididas pela variância, são dadas por:

$$\rho_1 = \frac{\gamma_1}{\gamma_0} = \frac{-\theta_1 \sigma_\varepsilon^2}{(1+\theta_1^2)\sigma_\varepsilon^2} = \frac{-\theta_1}{(1+\theta_1^2)} \quad (3.9)$$

A função de autocorrelação do modelo MA(1) apresenta apenas a primeira autocorrelação não nula e as demais iguais a zero. A primeira autocorrelação será positiva se θ_1 for menor que zero e negativa se θ_2 for maior que zero.

Modelos Mistos Auto-Regressivo de Médias Móveis (ARMA)

Em algumas situações faz-se necessário utilizar uma mescla dos componentes de um modelo AR com os componentes de um modelo MA, gerando assim, um modelo ARMA. O modelo ARMA(p,q) exigirá um número menor de termos e pode ser expresso conforme a seguinte equação:

$$\tilde{Z}_t = \emptyset_1 \tilde{Z}_{t-1} + \cdots + \emptyset_p \tilde{Z}_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \cdots - \theta_q \varepsilon_{t-q} \qquad (3.10)$$

A representação do modelo ARMA mais simples é o ARMA(1, 1), dado pela equação 3.11.

$$\tilde{Z}_t = \emptyset_1 \tilde{Z}_{t-1} + \varepsilon_t - \theta_1 \varepsilon_{t-1} \qquad (3.11)$$

A função de autocorrelação do modelo ARMA(1,1) é representada pela equação 3.12.

$$\rho_1 = \frac{(1-\emptyset_1\theta_1)(\emptyset_1\theta_1)}{1+\theta_1^2+2\emptyset_1\theta_1} \qquad (3.12)$$

Em que:

$\rho_k = \emptyset_1 \rho_{k-1}$, para $k > 1$.

Os modelos ARMA(p,q) apresentam características da função MA(q) em sua função de autocorrelação para as defasagens $k < q$, pelo fato de a memória do componente de médias móveis durar apenas q períodos. Para defasagens maiores que $k + 1$ as características são iguais às de um modelo AR(p).

3.2.2 Modelo Não Estacionário

As séries temporais que apresentam média e variância ao longo do tempo são conhecidas como não estacionárias. Nas séries não estacionárias existe uma inclinação nos dados e eles não permanecem ao redor de uma linha horizontal ao longo do tempo e/ou a variação dos dados não permanece essencialmente constante sobre o tempo, isto é, ao passar do tempo as flutuações aumentam ou diminuem, indicando alterações na variância. Para saber se uma série é não-estacionária, pode ser analisado gráfico da série, ou, então, aplicando os testes estatísticos de raiz unitária (WERNER e RIBEIRO, 2003).

Modelos Auto-Regressivos Integrados de Médias Móveis (ARIMA)

Quando $W_t = \Delta^d Z_t$ for estacionária, representa-se W_t por um modelo ARMA(p,q).

$$\emptyset(B)W_t = \theta(B)\varepsilon_t \tag{3.13}$$

Em que:

W_t é o resultado de diferenças de Z_t, ou seja, Z_té uma integral de W_t;

Assim se diz que Z_t segue um modelo auto regressivo integrado de médias móveis, ou modelo ARIMA(p,d,q) , onde p e q são as ordens de $\emptyset(B)$ e $\theta(B)$, respectivamente.

Assim com os modelos anteriores o modelo ARIMA(p, d, q) também pode ser reescrito utilizando o operador de defasagem B, em que

$$\left(1 - \emptyset_1 B - \cdots - \emptyset_p B^p\right) W_t = (1 - \theta_1 B - \cdots - \theta_q)\varepsilon_t$$

sendo $W_t = (1 - B)_d Z_t$. Então,

$$\emptyset(B)(1 - B)_d Z_t = \theta(B)\varepsilon_t. \quad (3.14)$$

Os modelos ARIMA resultam da combinação de três componentes denominados de filtros, o componente auto-regressivo (AR), o filtro de interação (I) e o componente de médias móveis (MA). Uma série pode ser modelada pelos três filtros ou apenas um subconjunto deles.

3.2.3 Modelos Sazonais

Os modelos ARIMA exploram a autocorrelação entre os valores da série em instantes sucessivos, mas quando os dados são observados em períodos inferiores a um ano, a série também pode apresentar autocorrelações para uma estação de sazonalidades. Os modelos que contemplam as séries que apresentam autocorrelação sazonal são conhecidos como SARIMA.

Esses modelos contêm uma parte não sazonal, com parâmetros (p, d, q), e uma parte sazonal, com parâmetros (P,D,Q). O modelo mais geral é dado pela seguinte equação:

$$\left(1 - \emptyset_1 B - \cdots - \emptyset_p B^p\right)\left(1 - \Phi_1 B^s - \cdots - \Phi_p B^{ps}\right)(1 - B)^d (1 - B^s)^D Z_t = \left(1 - \theta_1 B - \cdots - \theta_q B^q\right)(1 - \Theta_1 B^s - \cdots - \Theta_Q B^{Qs})\varepsilon_t \quad (3.15)$$

Em que:

$\left(1 - \emptyset_1 \mathrm{B} - \cdots - \emptyset_\mathrm{p} \mathrm{B}^\mathrm{p}\right)$ é a parte auto-regressiva não sazonal de ordem p;

$\left(1 - \Phi_1 \mathrm{B}^\mathrm{s} - \cdots - \Phi_\mathrm{p} \mathrm{B}^{\mathrm{ps}}\right)$ é a parte auto-regressiva sazonal de ordem P e estação sazonal s;

$(1 - \mathrm{B})^\mathrm{d}$ é a parte de integração não sazonal de ordem d;

$(1 - \mathrm{B}^\mathrm{s})^\mathrm{D}$ é a parte de integração sazonal de ordem D e estação sazonal s;

$\left(1 - \theta_1 \mathrm{B} - \cdots - \theta_\mathrm{q} \mathrm{B}^\mathrm{q}\right)$ é a parte não sazonal de médias móveis de ordem q;

$(1 - \Theta_1 \mathrm{B}^\mathrm{s} - \cdots - \Theta_\mathrm{Q} \mathrm{B}^{\mathrm{Qs}})$ é a parte sazonal de médias móveis de ordem Q e estação sazonal s;

Para Morretin e Toloi (2004), a estratégia para construção do modelo baseia-se em um ciclo iterativo, na qual a escolha da estrutura do modelo tem como base os próprios dados. Box e Jenkins (1970) apresentam três etapas para construção do modelo, que são:

1. Identificação: consiste em descobrir qual dentre os modelos de *Box-Jenkins* sejam eles sazonais ou não, descreve o comportamento da série. A identificação do modelo a ser estimado ocorre pelo comportamento das funções de autocorrelações (ACF) e das funções de autocorrelações parciais (PACF);

2. Estimação: Consiste em estimar os parâmetros $\emptyset$ e Φ do componente autoregressivo, os parâmetros θ e Θ do componente de médias móveis e a variância de ε_t;
3. Verificação: Consiste em verificar se o modelo estimado é adequado para descrever o comportamento dos dados.

3.3 Modelos de Filas

O estudo de A.K. Erlang em 1917, onde foram estabelecidas técnicas de probabilidades para determinar o número ótimo de linhas telefônicas e controlar as freqüências das chamadas telefônicas, foi uma das primeiras aplicações no estudo de filas. Hoje a teoria de filas é tema de estudo em várias área tais como: comunicação digital, hospitais, redes neurais, oficinas mecânicas, bancos, dentre outras. Basicamente o sistema de fila pode ser descrito como clientes chegando a um determinado serviço. Se o número de clientes que chegam para o atendimento for menor que o número de clientes que saem do sistema, o cliente aguarda o atendimento, após o atendimento o cliente deixa o sistema. A Figura 3.1 ilustra um sistema genérico de fila.

Figura 3.1: Sistemas de Filas.

Para a modelagem de qualquer fila, faz-se necessário calcular a taxa média de chegada dos clientes (λ), a taxa média de serviço(μ), identificar estatisticamente as correspondentes distribuições de probabilidade das mesmas, estabelecer o número de servidores disponíveis em cada instalação de serviço (s), e a ordem em que os clientes são retiradas da fila (disciplina da fila, geralmente do tipo primeiro a chegar, primeiro a ser atendido). Esses dados consistem nas entradas do processo de modelagem, cuja notação é a seguinte:

N(t) = Número de clientes no sistema de fila no tempo t (t>0);

λ = Taxa média de chegada;

μ = Taxa média de serviço;

NS = Número de servidores no sistema de fila;

Por sua vez, os referidos dados de saída, são:

L = Número de clientes esperado no sistema de fila;

T_s = Tempo médio atendimento dos clientes;

L_q = Número de clientes esperado apenas na fila;

W = Tempo de espera dos clientes no sistema (tempo de fila e em atendimento);

W_q = Tempo de espera dos clientes exclusivamente na fila;

P_n = Probabilidade de existirem exatamente n clientes no sistema de fila;

$\rho = \frac{\lambda}{NS * \mu}$, corresponde ao fator de utilização da instalação de serviço, ou seja a fração de tempo esperada em que os servidores estão ocupados.

3.3.1 Tipos de Filas

Para Maister (1995) os sistemas de filas são diferenciados pelo número de estágios e de processamento que possuem, podendo ser classificados como:

- Sistemas de único estágio: o sistema apresenta apenas um estágio de processamento, ou seja, os clientes passam somente por um atendente durante todo o processo. Esse tipo de sistemas é ilustrado na Figura 3.2.
- Sistemas Múltiplo estágio: os clientes devem passar por diversos estágios de processamento antes de deixarem o sistema. A Figura 3.3 ilustra esse sistema.

Figura 3.2: Sistema Único Estágio.

O número de filas que um cliente deve passar depende do número de estágios existentes no sistema. Normalmente o número de filas é igual ao número de estágios. Deste modo, pode-se, em conjunto com os tempos esperados de processamento, determinar o tempo esperado de permanência do cliente no sistema. A Figura 3.3 ilustra um sistema de múltiplos estágios.

Figura 3.3: Sistema Múltiplos Estágios.

Os sistemas de filas também podem ser classificados considerando o número de processadores/atendentes e a quantidade de filas (MAISTER, 1995). Sendo assim temos:

Sistema Único Estágio Paralelo: consiste em vários processadores dispostos em paralelo, cada um, precedido por uma fila. O Sistema Único Estágio Paralelo é ilustrado na Figura 3.4.

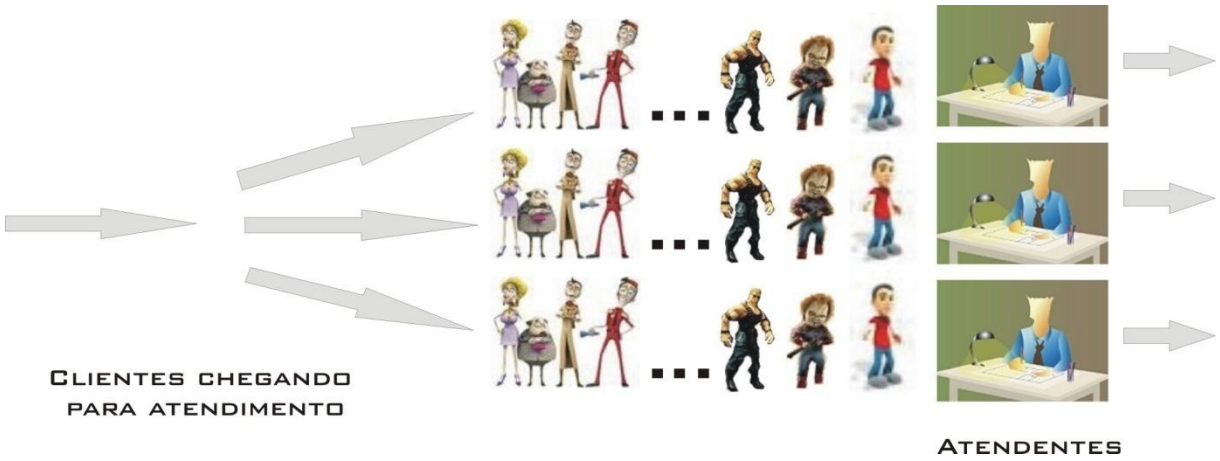


Figura 3.4: Sistema Único Estágio Paralelo.

Sistema Multicanal Único Estágio: Múltiplos processadores/atendentes são dispostos em paralelo precedidos por uma única fila. A Figura 3.5 ilustra esse tipo de sistema.

Sistemas Multi-filas: Um único processador/atendente é precedido por mais de uma fila, como em um semáforo em um cruzamento de vias. Esse sistema é ilustra na Figura 3.6.

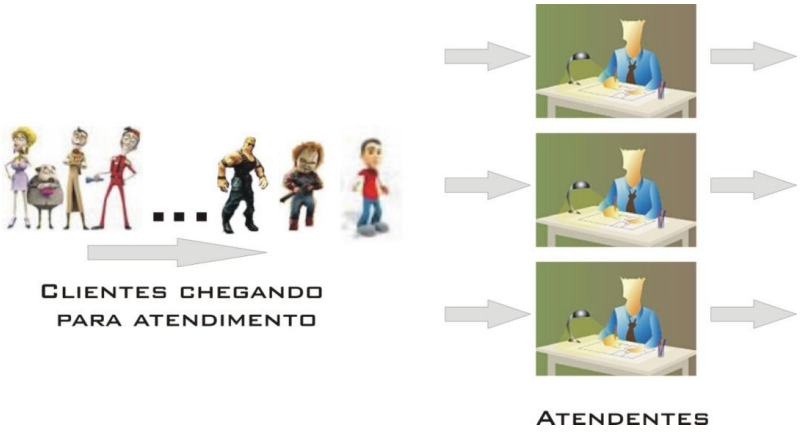


Figura 3.5: Sistema Multicanal Único Estágio.

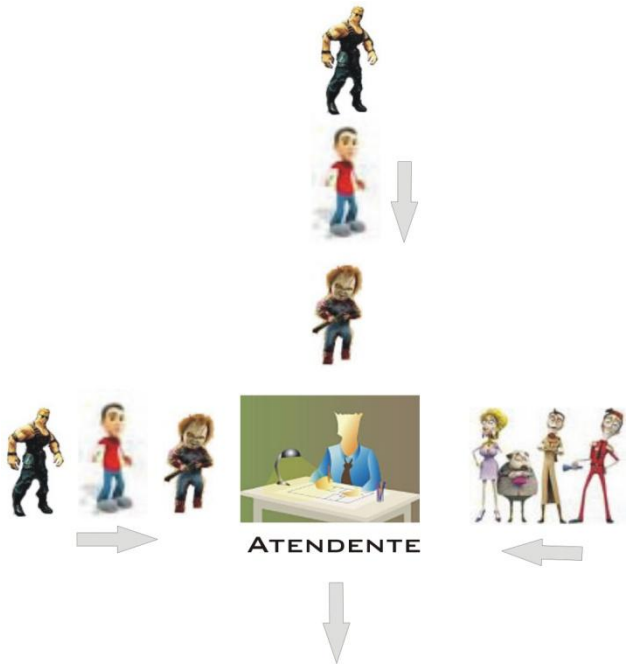


Figura 3.6: Sistema Multi-filas.

Sistema Discriminatório de Clientes: Os clientes são selecionados antes de serem encaminhados para as filas dos processadores, assim, cada processador se especializa no atendimento de um determinado tipo de consumidor, como em operações de *check-in* de aeroportos. A Figura 3.7 ilustra esse tipo de sistema.

Figura 3.7: Sistema Discriminatório de Clientes.

3.3.2 Notação de Kendall

Uma fila pode ser representada pela notação de Kendall, essa notação é uma forma geral de escrita como A/B/c/N/K/Z, onde:

- A - descreve a distribuição dos intervalos entre chegadas (símbolos comuns para A e B incluem M (exponencial ou Markoviano), D (constante ou determinístico), Ek (Erlang de ordem K), PH (fase - tipo), H (hiperexponencial), G (arbitrário ou geral), e GI (geral e independente));
- B- descreve a distribuição do tempo de serviço (símbolos comuns para A e B incluem M (exponencial ou Markoviano), D (constante ou determinístico), Ek (Erlang de ordem K), PH (fase - tipo), H (hiperexponencial), G (arbitrário ou geral), e GI (geral e independente));
- c- representa a quantidade de atendentes paralelos;
- N- representa a capacidade máxima do sistema (número máximo de clientes no sistema);
- K- é o tamanho da população;
- Z- é a disciplina da fila.

Os modelos mais comuns utilizados em sistemas de comunicação são:

- M/M/1 onde o primeiro “M” representa que a chegada de pacotes possui distribuição de Poisson, o segundo “M” representa que o tempo de serviço possui uma distribuição exponencial e o número “1” representa que somente um servidor está atendendo a fila;
- M/G/1 onde a chegada é de Poisson, denotada pelo primeiro “M”, a distribuição de tempo de serviço é genérica, denotada pela letra “G”, e apenas um servidor;

- M/D/1 onde o primeiro “M” representa que a chegada possui distribuição de Poisson, cujo o símbolo “D” representa que o tempo de serviço é fixo (constante), e “1” representa que apenas um servidor está atendendo a fila;
- M/M/m onde o primeiro “M” representa que a chegada de pacotes possui distribuição de Poisson, o segundo “M” representa que o tempo de serviço possui uma distribuição exponencial e “m” representa mais de um servidor atendendo a fila (SCHWARTZ, 1977,1992; BERTSEKAS e GALLAGER, 1992).

3.3.3 Disciplina das filas

A disciplina das filas compreende na forma pela qual os clientes que entram no sistema são atendidos, ou seja, uma lógica ordenada que determina como os clientes utilizarão os serviços quando um servidor estiver livre. Segundo Gross e Harris (1985), um sistema de fila é aquele em que algum produto flui, movimenta, ou é transferido através de um ou mais canais de capacidade finita para ir de um ponto para o outro. As disciplinas de fila mais comum são:

- FIFO (primeiro a chegar é o primeiro a ser atendido);
- LIFO (último a chegar é o primeiro a ser servido);
- SIRO (serviço de ordem randômica);
- SPT (tempo de baixo processamento primeiro); e
- PRI (serviço de acordo com a prioridade).

Note que a disciplina de filas FIFO, que será utilizada neste estudo, implica que os serviços são feitos de acordo com a ordem de chegada dos clientes, porém, isto não quer dizer que a ordem de chegada de um cliente será igual a ordem de saída pois às vezes os tempos de serviços requeridos são de diferentes tamanhos. Outra característica importante referente a fila de espera é a determinação da capacidade (*buffer*), que pode ser limitada ou ilimitada.

Sistema com Buffer Infinito

Neste tipo de sistema é considerado que a capacidade do buffer (área de espera) será sem restrições, ou seja, com relação a área de espera, o número de clientes aguardando serviço pode tender a infinito. Para um modelo de filas G/G/m são considerados as seguintes identidades.

O número de clientes no sistema é dado por:

$$L = \lambda W \tag{3.16}$$

e o número de clientes que esperam no buffer é:

$$L_q = \lambda W_q \tag{3.17}$$

Sistema com *Buffer* Finito

Um sistema com *buffer* finito é aquele que tem capacidade restrita. Supondo que o *buffer* comporta no máximo N clientes, se chegar um cliente e encontrar o *buffer* com a capacidade máxima N, então o cliente N + 1 não entra no sistema e a fonte será bloqueada. (Por outro lado se um cliente tiver grande necessidade de percorrer o sistema, este espera na fonte até que o sistema seja desbloqueado).

Considerando a importância das variáveis, medidas de congestionamento de tráfego, tempo gasto por um cliente no sistema ou o tempo de espera nos *buffers*, pode-se considerar as seguintes identidades descritas por Suri et al. (1993), para uma fila G/G/m/N. Seja λ_{eff} , a taxa com que os clientes entram no sistema. O número de clientes no sistema é dado por:

$$L = \lambda_{eff} W \tag{3.18}$$

E o número de clientes que esperam na fila é:

$$L_q = \lambda_{eff} W_q \tag{3.19}$$

O significado da taxa de chegada e λ_{eff} é relatado através da probabilidade do cliente chegar e encontrar o sistema cheio. Que é:

$$\lambda_{eff} = \lambda(1 - q_N) \text{ (G/G/m/N)} \tag{3.20}$$

Em que $(1 - q_N)$ é a fração de chegada que realmente entra no sistema. Se o processo de chegada for Poisson, então a probabilidade do tempo de chegada é $q_n = P_n$, lembrando que em Poisson utiliza-se as taxas médias dos tempos (WOL apud SURI et al. 1993).

Como a capacidade finita do sistema de filas é geralmente estável, partindo do princípio que os clientes são perdidos caso o limite permitido pela capacidade do sistema for ultrapassado. Então pode-se considerar que o tempo que um cliente gasta no sistema pode ser calculado pela seguinte relação:

$$L = L_q + \lambda_{eff}\tau \text{ (G/G/m/N)} \tag{3.21}$$

onde τ é o tempo gasto pelo servidor para atender um cliente.

Os fundamentos matemáticos apresentados neste capítulo são utilizados neste trabalho para fazer a previsão do volume de tráfego experimentado pelo *gateway* e calcular o número de processos destinados para atender dinamicamente cada classe de aplicação.

4 TRABALHOS RELACIONADOS

Neste capítulo abordaremos os trabalhos relacionados aos temas citados nos capítulos 2 e 3, dando ênfase aos trabalhos inerentes a este.

Autores como Lei et al. (2002) e Hwang e Tseng (2006) propõem arquiteturas para *Residential Gateways* (RGs) visando garantir qualidade de serviço. Em suas arquiteturas utilizam classificação de tráfego e administração racional da largura da banda disponível. Para cada tipo de aplicação os autores definem classes de QoS agrupando tais aplicações pela semelhança do tráfego gerado. Porém, todas as aplicações de tempo real são classificadas em uma mesma classe, não fazendo distinção entre *hard real time* e *soft real time*.

Schill et al. (1997) propõe uma aproximação de QoS para comunicação de RPCs em redes ATM (*Asynchronous Transfer Mode*), com total garantia de largura de banda para os hosts envolvidos. Este trabalho foca especificamente na comunicação constante entre dois hosts. A aproximação proposta usa a manutenção de comunicação entre dois hosts para diminuir o número de conexões lógicas estabelecidas e usa o tempo que seria gasto com a configuração de conexão, para a transmissão de RPCs (*Remote Procedure Call*). Porém, esse trabalho não considera as demais aplicações vigentes na rede.

Matschulat (2007) propõe um sistema de provisão de QoS em escalonadores para sistemas operacionais embarcados de tempo real. Em seu trabalho o autor se baseia no algoritmo *Reservation-based Earliest Deadline First* (R-EDF), e sugere modificações para melhorar o desempenho e adicionar suporte a aplicações *hard real-time.* Esse novo algoritmo é denominado *Enhanced Reservation-based Earliest Deadline First* (ER-EDF). Porém esse trabalho apresenta somente soluções para escalonamento de processos, não abordando o escalonamento de requisições feitas via rede.

Peixoto et al. (2008) propõe um modelo ortogonal para provisão de QoS aplicada a cluster de servidores Web. O modelo que utiliza uma disciplina de recursos original aliada à

disciplina de fila *Exigency-Based Scheduling* (EBS), onde o escalonador da fila está na horizontal e o escalonador de recursos na vertical, atuando assim, nessa visão, de forma ortogonal. O autor compara várias políticas de escalonamento para três classes de aplicações. Em seu trabalho o autor considera as aplicações de tempo real como uma única classe não considerando as peculiaridades dessas aplicações.

Wang et. al. (2005) propõe uma arquitetura de *proxy* do tipo *environment-aware* que provê QoS para um legado de aplicativos sem a necessidade de adicionar parâmetros para QoS. Com isso, a arquitetura permite a coexistência de legado de aplicações com aqueles que já possuem esses parâmetros. O autor apresenta uma aplicação gráfica onde os usuários do legado de aplicações podem especificar os requisitos de QoS de cada aplicação e, em seguida, enviar para o *proxy* através dos perfis de usuários. Como nos trabalhos anteriores, esta arquitetura não considera o tráfego de tempo real, considerando apenas o fluxo de aplicações convencionais (i.e. HTTP, FTP e multimídia).

Nobile (2007) apresentou uma arquitetura de *proxy* com QoS para RPCs-RT. O objetivo dessa arquitetura é privilegiar chamadas remotas de procedimento com maior prioridade e menor *deadline*, sem desconsiderar as demais chamadas com característica diferentes, com gerenciamento de recursos dinâmicos. Para gerir os recursos dinamicamente o autor propõe a utilização do modelo matemático *Box-Jenkins* que utiliza volumes anteriores para fazer previsão dos estados futuros de ocupação da rede e alocar, de acordo com políticas de atuação estipuladas, os recursos adaptando assim o *proxy* aos estados futuros do ambiente (NOBILE et al, 2007). A Figura 4.1 ilustra o modelo proposto.

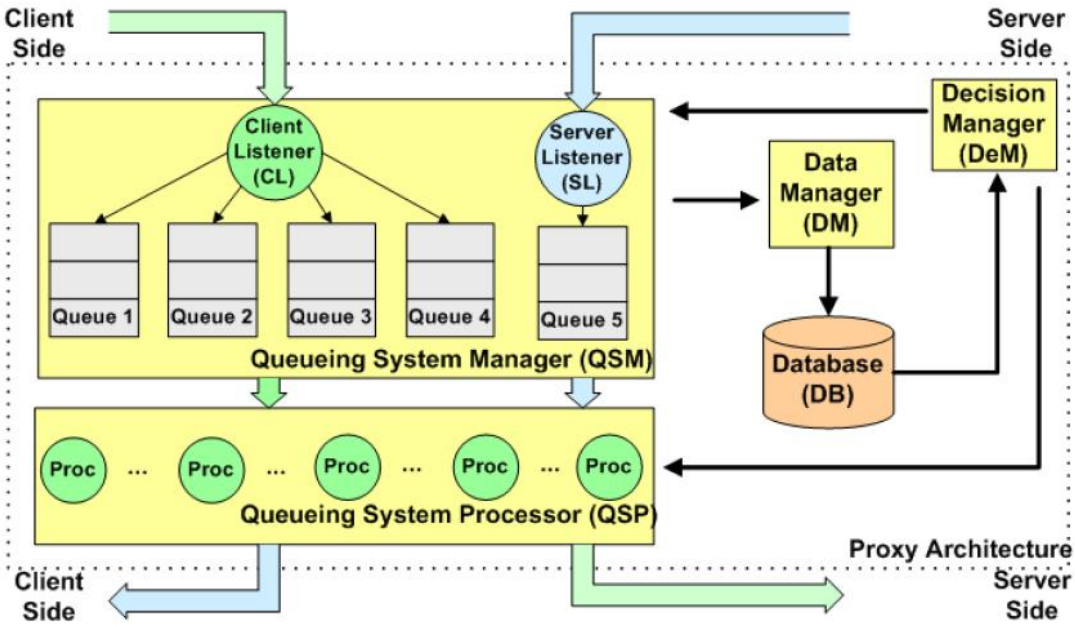


Figura 4.1: Arquitetura do Proxy com QoS para RPCs-RT (NOBILE, 2007).

O autor descreve os módulos expostos na Figura 4.1 da seguinte maneira:

O **Queuing System Manager (QSM)** é o módulo da arquitetura responsável por receber todo tráfego de mensagens que passa pelo *proxy* e organizá-las. Este módulo recebe todo tráfego tanto do lado do cliente quanto do lado do servidor e organiza-o em estruturas de fila chamadas *Queue*, numeradas de 1 (um) a 5 (cinco). Estas filas são independentes e podem apresentar comportamento distinto umas das outras.

O ***Client Listener*** **(CL)** recebe todo tráfego originado no *client side* e organiza em 4 (quatro) estruturas de fila. Cada estrutura de fila é responsável por um grupo de prioridades específico. Cada nova mensagem, que encapsula uma RPC-RT e chega ao CL, tem seu campo de prioridade verificado para identificar o grupo ao qual a mesma pertence. Além das mensagens RPCs, o CL ainda responsabiliza-se por organizar as mensagens que encapsulam pedidos de confirmação e cancelamento. Essas mensagens são inseridas na estrutura de fila do grupo de prioridade baixa, com a política *First Come First Served* (FCFS).

O ***Server Listener*** **(SL)** é responsável pelo tráfego que entra no *proxy* pelo *server side*. Estas são mensagens do tipo *Promise*, ou seja, armazenam os valores resultantes da execução

de uma RPC-RT. As mensagens do tipo *Promise* não possuem prioridades ou requisitos temporais, assim como as mensagens de confirmação e cancelamento, o que permite a utilização de uma única estrutura de fila, a *Queue* 5 (cinco), para organizá-las.

O ***Queueing System Processor* (QSP)** é responsável pela extração das mensagens organizadas e inseridas nas estruturas *Queue* do QSM e pelo direcionamento para linha de saída correta, seja ela o *server side* ou o *client side*. Cada processador é capaz de processar um número limitado de mensagens por unidade de tempo, e sendo assim, quanto maior o número de mensagens, maior deve ser a quantidade de processadores alocados suficientes para atender a demanda atual.

O ***Data Manager* (DM)** recebe as informações sobre estatísticas arrecadadas pelo QSM, sumariza e armazena nas estruturas adequadas para persisti-las no *Database* (DB). Essas informações devem estar disponíveis para o acesso do *Decision Manager* que as utiliza para o processo de tomada de decisão. Por meio da ação do DM que o *Decision Manager* obtêm informações sobre os estados passados do ambiente de rede, uma vez que estas representam o volume do tráfego e as características das mensagens que passaram pelo *proxy*.

O ***Decision Manager* (DeM)** determina a reação dos demais módulos aos estados do ambiente de rede, por meio da escolha de políticas de comportamento. O processo de tomada de decisão envolve o entendimento dos estados anteriores e do estado atual experimentados, e dos recursos disponíveis para o *proxy* atender a demanda de tráfego. As decisões tomadas pelo DeM são repassadas aos módulos correspondentes para determinar o comportamento dos mesmos em relação aos estados futuros. Esse módulo é composto pelo *Prediction System* (PS) e *Decision System* (DS).

O ***Prediction System*** realiza previsões com o intuito de antecipar os estados futuros do ambiente de rede. Com base nos estados passados (i.e. volume e característica das mensagens RPCs-RT que passaram pelo *proxy*), o PS utiliza modelos de séries temporais (*Box-Jenkins*)

adequados para realizar previsões sobre a características e a quantidade de mensagens RPCs-RT para qual o *proxy* deve estar preparado. As previsões fornecem base às tomadas de decisões e permitem que o *proxy* adeque-se a um futuro breve do seu ambiente. Em intervalos de tempo bem definidos, o PS recupera as informações, outrora, armazenadas na *Database* (DB), para realizar suas previsões.

O **Decision System (DS)** é o administrador da tomada de decisões do *proxy*. Dos módulos do *Decision Manager*, o DS executa o papel fundamental ao qual se propõe o DeM, uma vez que recebe os dados de entrada, executa a tomada de decisão e repassa a mesma aos demais módulos do *proxy*. As decisões envolvem diretamente a alocação racional dos recursos do *proxy* e escolha de políticas de comportamento para os módulos QSM e QSP.

Em seu trabalho, Nobile (2007) insere as tarefas nas filas no sistema *First Come First Served* não verificando a prioridade das tarefas que já estavam na fila. Outra característica é que as prioridades das tarefas são divididas em quatro grupos, restringindo as tarefas a um grupo por fila. Os processadores responsáveis por uma mesma fila seguem políticas idênticas de extração. O número de processadores foi limitado em 1024 e só poderão ser realocados na próxima verificação do DS.

5 GATEWAY COM PREVISÃO DE TRÁFEGO PARA PRIORIZAÇÃO DE APLICAÇÕES DE TEMPO REAL

O objetivo do *gateway* é privilegiar a transmissão dos protocolos que possuem requisitos temporais, de maneira eficiente e racional sem desconsiderar os demais protocolos. Assim o *gateway* foi idealizado como uma arquitetura adaptável que gerencia os recursos disponíveis de maneira inteligente e equilibrada.

Para alocar os recursos com maior eficiência o *gateway* analisa o ambiente de rede em que está inserido, coleta os dados relevantes ao sistema e classifica os diferentes tráfegos de acordo com as características das aplicações vigentes na rede. Com base nos dados coletados, é possível prever o fluxo de tráfego de cada classe de serviço em instantes futuros, utilizando séries temporais, e auto-ajustar os recursos disponíveis momentos antes da chegada dos pacotes para atender a demanda.

Mesmo com o mecanismo de previsão e alocação antecipada de recursos, devem ser utilizadas políticas de comportamento para garantir que os recursos sejam disponibilizados de forma otimizada para cada classe de serviço visando atender as requisições antes do término de seus *deadlines*. As políticas interferem diretamente na eficiência do *gateway*, pois são responsáveis por determinar como será feita a distribuição dos recursos disponíveis para o atendimento das filas e conseqüentemente de cada classe de serviço. A escolha de boas políticas auxilia no aumento do desempenho e evita o desperdício de recursos. A Figura 5.1 ilustra a arquitetura do *gateway* proposta nesse trabalho.

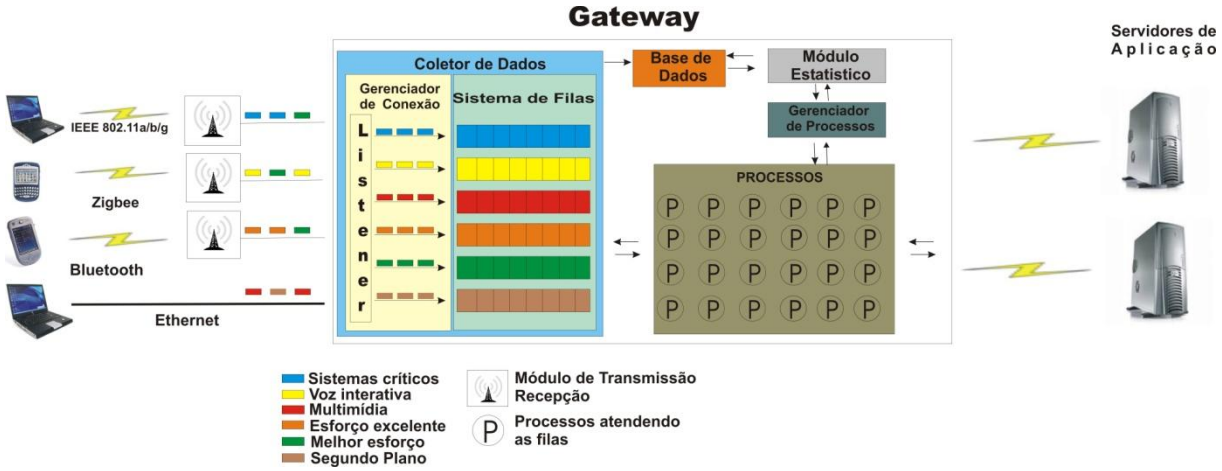


Figura 5.1: Arquitetura de Gateway Proposta.

A arquitetura proposta, ilustrada na Figura 5.1, é baseada no trabalho de Nobile (2007), que utilizou a metodologia *Box-jenkins* para prever o tráfego de chamadas RPC-RT e alocar dinamicamente os recursos do sistema. No decorrer do texto será explicitado os pontos em que fora utilizado recursos propostos por Nobile.

O *gateway* é organizado em módulos, cada módulo executando tarefas bem definidas a fim de aperfeiçoar o atendimento das requisições privilegiando as classes de serviços que possuem requisitos temporais. A seguir detalharemos esses módulos e suas funcionalidades.

5.1 Coletor de Dados (CD)

A função do módulo Coletor de Dados no sistema é contabilizar informações referentes ao tráfego de requisições que passam pelo *gateway* e armazená-las na Base de Dados para serem utilizadas posteriormente pelo Módulo Estatístico. As informações coletadas por esse módulo são:

- Número de requisições que chegam na fila;

- Número de requisições que saem da fila;
- Instante (tempo) de chegada de cada requisição;
- Número de requisições rejeitadas por falta de espaço na fila;
- Instante (tempo) em que cada requisição é retirada da fila;
- Número total de requisições que passaram pelo *gateway* até o momento;
- Quantidade de mensagem que perderam seu *deadline*;

O CD permanece ativo durante todo o tempo de execução do *gateway* e grava as informações coletadas momentos antes do Módulo Estatístico (ME) efetuar seus cálculos. Isso é feito por duas razões: primeiramente o tempo de acesso em memória é menor que o tempo de acesso a disco. A segunda razão é que os dados só serão utilizados em períodos determinados de tempo não havendo necessidade consumir recurso para a gravação de dados a todo o momento.

O CD captura as informações do módulo Gerenciador de Conexão e do módulo Sistema de Filas. Por essa razão na Figura 5.1 esses módulos estão representados dentro do Coletor de Dados.

5.1.1 Gerenciador de Conexão (GC)

Este módulo é responsável por gerenciar as conexões dos clientes, identificar a classe de serviço dos pacotes e encaminhá-los para a devida fila. O *Listener* é um sub-modulo do GC que fica aguardando as conexões dos clientes. Quando uma conexão é recebida o *Listener* gerencia os descritores até o término da conexão. O GC também é responsável por encaminhar os pacotes para as filas de acordo com as classes de serviço reconhecidas pelo *gateway*.

Classe de Serviço

A maioria das aplicações tradicionais da Internet, tais como e-mail ou transferência de arquivos por FTP, são sensíveis à perda de pacotes, mas podem tolerar atrasos. No caso de aplicações multimídia (voz e vídeo), estas toleram a perda de alguns pacotes, mas são sensíveis a atrasos e a variações do atraso (*jitter*). Devido as características de cada tipo de aplicação podemos classificá-las em grupos ou classes e tratá-las de forma diferenciada a fim de melhorar a qualidade do serviço disponível na rede.

Existem diversas abordagens a fim de melhorar a transmissão de pacotes na rede, uma delas é o padrão IEEE 802.1p, que é uma técnica para priorização de tráfego em redes locais, sendo especificado na norma IEEE 802.1D – LAN *Bridges*. Através dessa técnica, é possível utilizar aplicações sensíveis a tempo em redes locais (LANs). A classificação das aplicações bem como suas prioridades do padrão IEEE 802.1p são descritas na tabela 5.1:

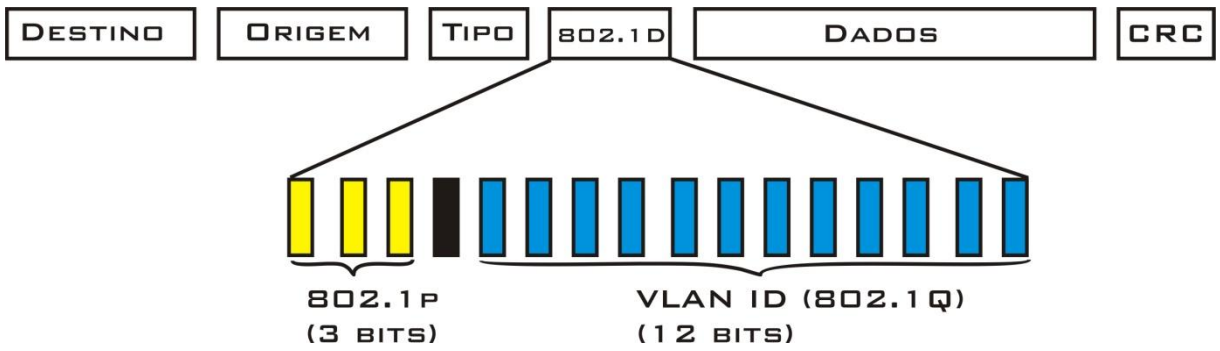
Tabela 5.1: Classificação segundo o padrão IEEE 802.1p.

Prioridade	Nome	Características e exemplos
7	Controle da rede	Aplicações críticas (RIP, OSPF, BGP4)
6	Voz interativa	Sensíveis à latência e jitter com baixa banda (Netmeeting, RAT)
5	Multimídia interativa	Sensíveis ao jitter com alta banda (picturetel, indeo)
4	Carga controlada	Aplicações sensíveis à latência (transações SNA)
3	Esforço excelente	Tráfego crítico que tolera atrasos (SAP, SQL)
2	Melhor esforço	Melhor esforço
1	Default	Default
0	Background	Insensíveis à latência (FTP, backups, pointcast)

Pode-se observar na tabela que aplicações de voz interativa têm prioridade maior que aplicações multimídia (*streaming* de áudio e vídeo), isso se deve ao fato das aplicações de voz

suportarem um *delay* menor que 10 milissegundos e aplicações multimídia suportarem um *deley* de 100 milissegundos.

A norma IEEE 802.1D inclui as extensões 802.1p e 802.1Q, adicionando 4 bytes ao formato do cabeçalho MAC das redes Ethernet e Token Ring. A extensão 802.1Q utiliza dois bytes desse espaço, sendo que 12 bits são reservados para identificação de VLAN (Virtual LAN) e 3 bits são definidos pela norma IEEE 802.1p a fim de determinar a prioridade no nível 2 do modelo OSI, conforme mostra a Figura 5.2:



.

Figura 5.2: Extensão do 802.1D.

Assim essa técnica fica limitada a enlaces locais (camada de enlace do modelo OSI). Uma forma de resolver esse problema é trabalhar em camadas superiores do modelo OSI. A arquitetura de *gateway* ora proposta, trabalha nas camadas superiores do modelo OSI sendo assim, não é necessário estender os cabeçalhos, mas sim classificar o fluxo de mensagem de acordo com as aplicações vigentes na rede. A classificação para os serviços propostos nesse trabalho é semelhante a proposta pelo padrão IEEE 802.1p, como podemos ver na tabela 5.2:

Tabela 5.2 Classe de serviço proposta para o gateway.

Prioridade	Descrição da classe	Características
6	Sistemas críticos	Essa classe engloba aplicações de tempo real crítica tais como controle de robôs em indústrias e controle de tráfego aéreo.

5	Voz interativa	Nesta classe temos aplicações de tempo real não crítica sensíveis à latência com *delay* abaixo de 10 milissegundos tais como Netmeeting, RAT e Skype.
4	Multimídia	Aplicações de tempo real não críticas com *delay* abaixo de 100 milissegundos tais como monitoramento online, streaming de áudio e vídeo.
3	Esforço excelente	Tráfego crítico que tolera atrasos como sistemas de gerenciamento, banco de dados e acesso remoto.
2	Melhor esforço	Aplicações pouco sensíveis a latência e jitter tais como Navegadores, Messenger, e-mail etc.
1	Segundo Plano	Aplicações insensíveis à latência como transferência de arquivos e backups.

Podemos notar algumas diferenças em relação ao protocolo IEEE 802.1p quanto à classificação das mensagens e protocolos. O *gateway* não considera os protocolos RIP, OSPF e BGP4, pois esses protocolos são especialmente utilizados por roteadores para atualizar as tabelas de roteamento e não passam pelo *gateway*.

Assim a classificação proposta visa abranger os protocolos da camada de aplicação e priorizar-los de acordo com sua sensibilidade a latência, tolerância a atrasos e *jitter* e *deadline.*

5.1.3 Sistema de Filas (SF)

O SF é o módulo responsável pelo gerenciamento das filas do sistema. Este módulo é composto por seis filas distintas. Cada fila recebe as requisições do GC já classificadas e armazena essas requisições em ordem de chegada até a retirada pelos Processos. Assim cada fila recebe o tráfego referente a uma classe de serviço organizando as requisições por grupos de prioridade evitando o trabalho dos Processos de percorrer as filas, verificado a prioridade de cada requisição como é feito com apenas uma fila, pois essa abordagem demanda maior tempo de processamento na retirada das requisições dos clientes da fila aumentando a latência gerada pelo sistema.

A tabela 5.3 representa a distribuição das classes de serviços por fila do *gateway* em ordem de prioridade.

Tabela 5.3: Distribuição das classes de serviço por fila.

Fila	Prioridade	Descrição da classe
5	6	Sistemas críticos
4	5	Voz interativa
3	4	Multimídia
2	3	Esforço excelente
1	2	Melhor esforço
0	1	Segundo Plano

5.2 Base de Dados (BD)

Na Base de Dados ficam armazenadas todas as informações coletadas pelo CD, as estatísticas geradas pelo Módulo Estatístico e as políticas implementadas pelo Gerenciador de Processos (GP). Toda a informação capturada pelo CD é armazenada pelo BD e é utilizada como base para os cálculos estatísticos gerados pelo ME que os armazena no BD. Após o ME gerar os dados o GP utiliza esses informações para definir o número de processos alocados por fila ou classe de serviço.

5.3 Módulo Estatístico (ME)

O Módulo Estatístico gera os dados estatísticos do sistema com base nos dados coletados pelo CD. Para cada fila, que representa uma classe de serviço, o ME calcula:

- Taxa de Chegada (λ) = $\frac{Total\ de\ requisições\ que\ chegaram\ no\ sistema}{Tempo\ de\ Execução}$;

Onde, *Tempo de Execução* corresponde ao tempo decorrido até o instante do calculo.

- Tempo médio de serviço (s) = $\frac{Tempo\ de\ Serviço}{Total\ de\ requisições\ que\ deixarm\ o\ sistema}$;

Onde, *Tempo de Serviço* corresponde a somatória do tempo gasto para atender as requisições até o instante do calculo.

- Taxa de serviço (μ) = $\frac{1}{s_i}$;
- Carga ou fator de utilização (ρ) = $\frac{\lambda}{\mu}$;
- Tempo médio de requisições na fila (W) = $\frac{Tempo\ total\ de\ ocupação\ da\ Fila}{Número\ de\ requisições\ que\ deixaram\ as\ filas}$;
- Tamanho médio da fila (nq) = $\frac{\lambda}{W}$;

Este módulo também é responsável por realizar as previsões com o intuito de antecipar os estados futuros da rede. As previsões são feitas com base nos estados passados, ou seja, no volume de requisições de cada classe de serviço que passaram pelo *gateway*. Para realizar as previsões o ME utiliza modelos matemáticos. As previsões geradas pelo ME são utilizadas pelo Gerenciador de Processos como base para a tomada de decisão permitindo ao *gateway* adequar-se a um futuro breve do ambiente, destinando recursos necessários para atender cada classe de serviço.

Em períodos regulares de tempo o ME efetua os cálculos estatísticos com as informações contidas na base de dados. Para cada fila é realizada uma previsão e com base no fluxo de mensagens que passaram pela fila. As previsões são feitas utilizando métodos conhecidos como séries temporais. Esses métodos partem do princípio que observações sucessivas do ambiente em estados passados e suas correlações podem ajudar a prever possíveis estados futuro. O método escolhido neste trabalho foi o método *Box-Jenkins* (Box e

Jenkins, 1976). Os modelos de *Box-Jenkins* partem da idéia de que cada valor da série (temporal) pode ser explicado por valores prévios, a partir do uso da estrutura de correlação temporal que geralmente há entre os valores da série.

Em seu trabalho Nobile (2007) propôs calcular a previsão em períodos regulares. Para isso, a série temporal deve ser modelada de acordo com as características do ambiente de rede. A modelagem consiste em calcular as funções de auto-correlação e auto-correlação parcial. Porém, em sua simulação, Nobile fez cada etapa separadamente, isto é, primeiro submeteu somente a parte do *proxy* destinada a coleta de dados ao tráfego gerado, depois calculou as funções de auto-correlação e auto-correlação parcial para estimar a ordem (p,d,q) da série e efetuar a previsão aplicando a metodologia *Box-Jenkins.*

O *gateway* ora proposto utiliza o modelo de previsão proposto por Noblie (2007) com algumas alterações. O *gateway* não precisa ser submetido previamente ao volume de tráfego para estimar a ordem (p,d,q) do modelo *Box-jenkin.* Para obter dados, o *gateway* implementa o de *warming up*, que será discutido posteriormente. Além disso, utilizou-se o pacote estatístico *R* (RProject, 2007) para o cálculo da previsão. Esta ferramenta permite criar funções para automatizar o processo de previsão, a partir de uma série temporal de entrada. Assim todo o processo de previsão no presente trabalho foi realizado através de uma interface criada entre o *gateway* e o pacote *R.* A interface permite que o ME execute uma função criada no *R* para calcular a previsão a partir dos dados presentes na BD e obter de volta o resultado. Em seguida o ME grava esses dados na BD para que esses possam ser utilizados pelo Gerenciador de Processos.

Para determinar a ordem (p,d,q) para o modelo ARIMA, fora utilizado a função auto.Arima da biblioteca *forecast* do R. Assim é estimada a melhor ordem para a série a ser analisada.

```
1  bj<-function(arquivo,coluna,destino){
2    library(forecast)
3    dados<-read.csv(arquivo,header=TRUE,sep=";")
4    x<-colSums(dados)
5    if(x[coluna] != 0)
6    {
7      dados<-ts(dados[,coluna])
8      fit1<-auto.arima(dados)
9      y<-forecast(fit1,h=5)
10     write.csv(y[4], file = destino, append = TRUE)
11   }
12   else
13   {
14      x <- c(0,0,0,0,0)
15      x1 <- ts(x)
16      write.csv(x1, file = destino, append = TRUE)
17    }
18 }
```

Figura 5.3: Função de previsão criada no R.

A Figura 5.3 demonstra a função criada no R para efetuar as previsões. A linha 2 carrega a biblioteca *forecast*. A linha 3 carrega os dados observados para a variável dados. A linha 7 converte os dados observados em série temporal. A linha 8 calcula a ordem (p,q,d) para o modelo ARIMA. A linha 9 efetua a previsão e retorna 5 (cinco) valores previstos para os próximos 5 (cinco) períodos.

5.4 Gerenciador de Processos (GP)

As políticas de comportamento do *gateway* são as diretrizes que devem ser tomadas pelos Processos. Essas políticas determinam como deve ser feito o escalonamento das requisições das filas visando aperfeiçoar os recursos disponíveis e evitar que requisições de

tempo real percam seus *deadlines*. O GP é responsável por auto-ajustar o sistema mediante as previsões feitas pelo ME.

Quando o *gateway* é iniciado a BD não contém dados para que o ME possa fazer as previsões, pois não houve nenhum fluxo de dados passando pelo *gateway*. Esse período inicial do sistema é chamado de *warming up*. Durante o *warming up* o GP pode ser configurado para atender as filas seguindo as seguintes políticas de escalonamento:

- *Round-Robin;*
- *First come, first served;*
- Prioridade;
- *Deadline;*

Outra abordagem para o período de *warming up* é distribuir os processos uniformemente para todas as filas, dividindo o número de processos disponíveis pelo número de filas no sistema. Para que a distribuição seja uniforme, o número de processos deve ser múltiplo do número de filas. Esta abordagem não considera os *deadlines* das requisições, podendo ser inadequado em situações em que o fluxo de mensagens de tempo real é muito alto.

Após o período *warming up*, o GP já tem informação para tomada de decisão baseada nas previsões feitas pelo ME e pelo número médio de requisições na fila atual. Assim o GP calcula o número de processos que serão disponíveis para cada fila da seguinte forma:

$$QP = \frac{(Taf + Tfp) * Ts}{D} \tag{5.1}$$

Em que:

- QP é o número de processo a ser alocado para a fila;

- Taf o tamanho médio da fila atual;
- Tfp tamanho médio da fila previsto;
- Ts tempo médio de serviço;
- D *deadline* da requisição;

A equação 5.1 foi proposta por Nobile (2007) em sua arquitetura de *proxy* e utilizada neste trabalho para determinar o número de processos a serem alocados para cada classe de aplicação. Com o número de processos definidos o GP determina aos processos quais filas cada processo deve atender. Os processos ficam alocados para o atendimento das filas até o próximo cálculo ou até a fila estar vazia. Nesta configuração as filas de prioridade mais altas passam a ficarem vazias mais rapidamente por ter um número maior de processos destinados ao seu atendimento. Assim que a fila fica vazia os processos verificam qual fila ainda não está vazia e então ajuda no atendimento desta fila momentaneamente, até a fila a qual o processo está alocado voltar a ter alguma requisição a ser atendida. A verificação é sempre feita da fila de maior prioridade para a fila de menor prioridade. Caso nenhuma fila tenha requisições para serem atendidas, os processos entram em estado de espera.

5.5 Processos

Este módulo contém os processos responsáveis por retirar as requisições das filas e direcionar para as linhas de saída. Como todos os processos podem atender qualquer fila dependendo das políticas determinadas pelo GP, os processos devem ser capazes de atender todas as classes de serviço vigentes na rede. O número de processos disponível no GP depende da capacidade do *hardware* em que o *gateway* foi instalado. O dimensionamento do

número máximo de processos suportado por um determinado *hardware* não é o foco deste trabalho e não é abordado no mesmo.

A seguir são apresentados os resultados da exposição do protótipo do *gateway* a um tráfego gerado com as diversas classes de serviço.

6 IMPLEMENTAÇÃO E TESTE

Neste capítulo é apresentada uma implementação da arquitetura de *gateway* proposta a fim de validá-la para os diferentes tipos de tráfego existentes na rede. Neste estudo, uma implementação da arquitetura é exposta a um volume variável de tráfego composto pelos tipos de requisições propostos para a arquitetura, com diferentes prioridades e *deadlines*, submetendo o *gateway* a maior quantidade e diversidade de estados possíveis.

Para realização dos testes foi desenvolvida uma aplicação responsável por gerar os diversos tipos de requisições com diferentes características. Esta aplicação gera volumes variáveis de requisições com requisitos temporais bem como requisições dos demais tipos de tráfego, os quais não possuem requisitos temporais, resultando em diferentes estados de tráfego na rede, com o intuito de inundar o *gateway* com diferentes volumes de requisições, criando um ambiente onde possam ser analisados os comportamentos dos diversos módulos da arquitetura, principalmente, os módulos envolvidos com a tomada de decisão.

O objetivo deste estudo é analisar se quando submetido a situações de estresse em que o volume de requisições que chegam supera a capacidade de atendimento imediato, o *gateway* realiza boas previsões sobre situações futuras e aloca os processos de forma ótima.

Como objetivo secundário, busca-se comparar os resultados obtidos pela utilização de diferentes políticas. A análise da aplicação das diferentes políticas de comportamento é feita por meio do controle das variáveis envolvidas.

Por variáveis envolvidas, entende-se as variáveis de controle da aplicação teste com relação a quantidade e características das requisições e as varáveis que controlam as ações do *gateway*. Nas próximas subseções serão detalhadas as etapas envolvidas neste estudo.

6.1 Desenvolvimento do Ambiente de Teste

O ambiente de teste desenvolvido é composto de dois módulos com funcionalidades distintas: o módulo gerador e o módulo receptor. O módulo gerador encarrega-se da geração das mensagens e da marcação do tipo de mensagem, enquanto que o módulo receptor é responsável por receber as mensagens geradas e encaminhar a resposta ao gerador.

O módulo gerador possui parâmetros que permitem controlar o número de requisições, bem como os intervalos de envio para cada classe de aplicação. Além disso, o módulo gerador marca cada requisição antes do envio para que possam ser identificadas pelo *gateway*.

Como o foco principal dos testes é analisar a validade da arquitetura aplicada a diferentes classes de aplicações simultaneamente e não o funcionamento dos protocolos, as requisições geradas pelo módulo gerador são bastante simples. As requisições foram capturadas da rede utilizando o *software* gratuito *Wireshark*, assim as requisições geradas possuem as mesmas características das requisições encontradas na rede. Para cada classe diferente, foi utilizado um *software* gratuito que trabalha com protocolos da classe em questão para a captura da requisição. Já no receptor, a resposta é simples, confirmando o recebimento.

A tabela 6.1 descreve os *softwares* utilizados para captura das requisições por classes de serviços.

Tabela 6.1: *Softwares* utilizados para captura de requisições.

Descrição da classe	***Software***
Sistemas críticos	OPC (OLE *for Process Control*)
Voz interativa	*Skype*
Multimídia	*RealPlayer*
Esforço excelente	*Real VNC*
Melhor esforço	*MSN Messenger*
Segundo Plano	*FileZilla*

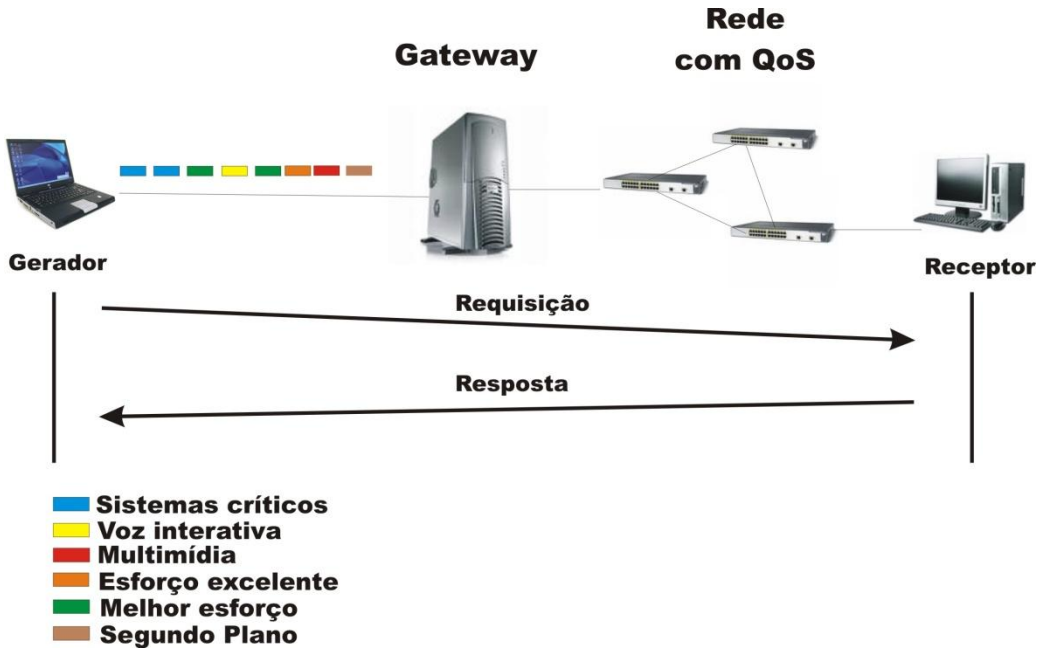


Figura 6.1: Ambiente de Teste.

Para realização dos testes, os módulos gerador e receptor foram implementados em linguagem C.

6.2 Módulo de Previsão

As previsões são realizadas por modelos estatísticos de séries temporais que utilizam informações sobre o volume de tráfego experimentado pelo *gateway* em instantes anteriores, armazenadas na BD. Para cada classe de serviço é identificado um modelo de série temporal para previsão.

Para que se possam fazer previsões é necessário ter um número de observações em um determinado período de tempo. Inicialmente o *gateway* não possui nenhuma informação sobre o volume de tráfego da rede. Assim faz-se necessário um período inicial para a coleta de informações que chamaremos de *warming up*. Durante esse período o *gateway* armazena o

número médio de requisições na fila em intervalos discretos. Isso é feito para todas as filas do sistema. Passado o período de *warming up* o *gateway* já possui um histórico de observações para cada fila.

Para determinar os modelos de previsão, as informações sobre o fluxo de requisições são organizadas em séries para que sejam aplicados os passos do modelo matemático *Box-jenkins* visto na seção 3.2, sendo o processo de modelagem de comportamento da rede análogo para cada classe de serviço vigente na rede.

Os passos da modelagem envolve o cálculo das funções de auto-correlação e auto-correlação parcial, e a estimativa de parâmetros. Para tais cálculos, foi utilizada a versão 2.8.1 do pacote de ferramentas estatísticas e matemáticas R (RProject, 2007) e a biblioteca *Forecasting*, disponível para a ferramenta R.

A interação entre o *gateway* e o R é feita através de uma interface criada para tornar possível o cálculo das previsões em tempo de execução. A interface consiste de funções desenvolvidas no R para cálculo das previsões. Primeiramente é feita a conversão dos valores observados em séries temporais, depois é estimado os parâmetros do método ARIMA com a função da biblioteca *Forecasting* Auto.Arima() e finalmente é calculada as previsões com a função Arima().

Em intervalos regulares o *gateway* executa uma chamada remota ao R passando como parâmetro a fila e os valores das observações feitas. O pacote estatístico R executa as previsões de acordo com funções criadas e retorna os valores previstos. Então o *gateway* passa a calcular o número de processos necessários para atendimento das filas.

6.3 Módulo de Previsão para Requisições de Sistemas críticos, Voz interativa e Multimídia

O modelo ora proposto tem por finalidade privilegiar as requisições que possuem menor *deadline* conseqüentemente prioridade mais alta. Para sistemas com essa característica, a solução ideal seria redirecionar as requisições imediatamente, pelo *gateway*, da linha de entrada para a linha de saída correta. Contudo, em sistemas sem controle de tráfego, as requisições de alta prioridade com requisitos rígidos de tempo concorrem igualmente com outras mensagens de menor prioridade, gerando descarte de requisições por perda de *deadline*.

Para diminuir o descarte de requisições de maior prioridade por perda do *deadline*, a arquitetura proposta visa manter a ocupação da fila de alta prioridade sempre com a menor ocupação possível, encaminhando essas requisições mais rápido que as demais. Para privilegiar as filas de maior prioridade o *gateway* utiliza previsões que são utilizadas como base na tomada de decisão. Tais previsões informam a carga esperada para instantes futuros baseadas em observações feitas anteriormente.

Uma modelagem precisa dos valores já observados possibilita previsões muito próximas da realidade. Para minimizar o desvio entre os valores previstos em relação aos valores reais, para toda previsão, só são considerados os valores reais observados, sendo assim desconsideradas as previsões. Mesmo um modelo adequado pode produzir previsões que não refletem a realidade das observações. A seguir é ilustrada uma comparação entre os valores reais observados e as previsões realizadas pelo *gateway* para as filas que possuem alguma restrição temporal.

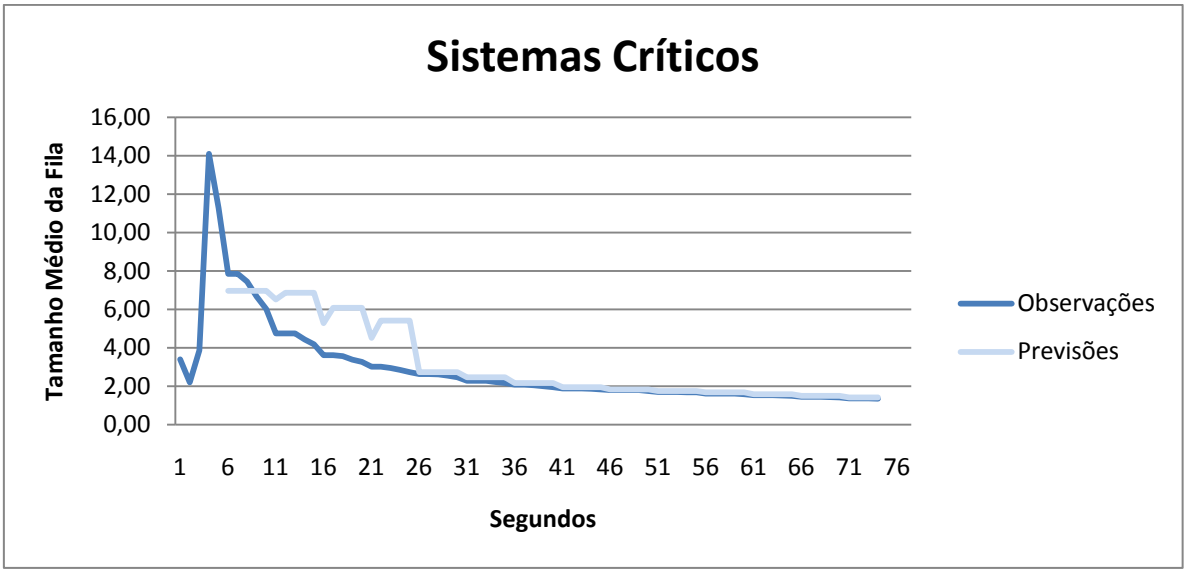


Figura 6.2: Previsão X Observações para o tráfego de sistemas críticos.

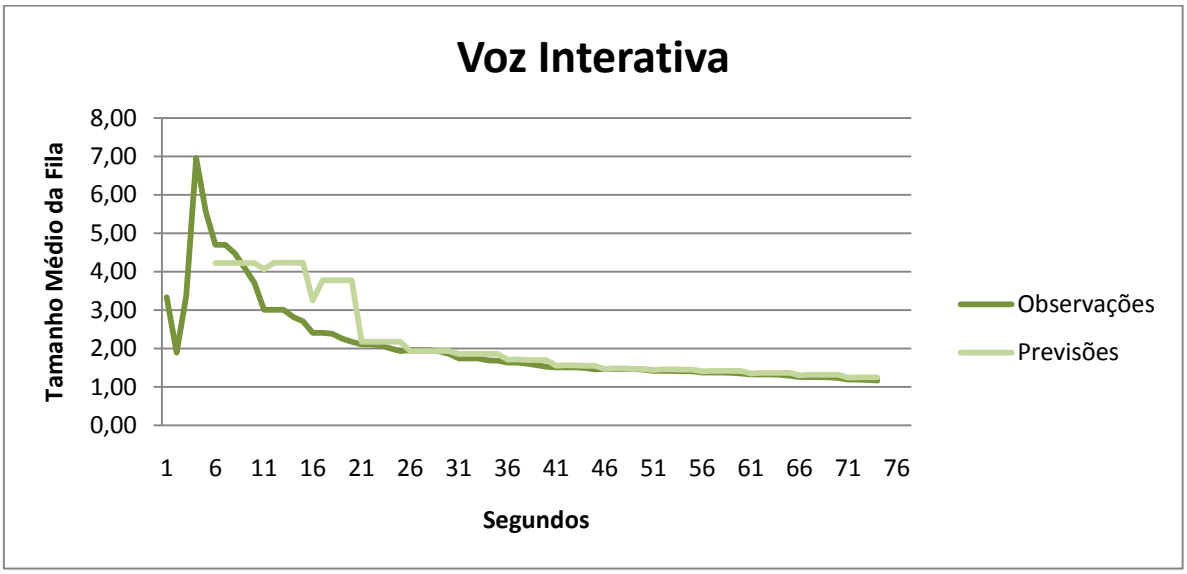


Figura 6.3: Previsão X Observações para o tráfego de Voz Interativa.

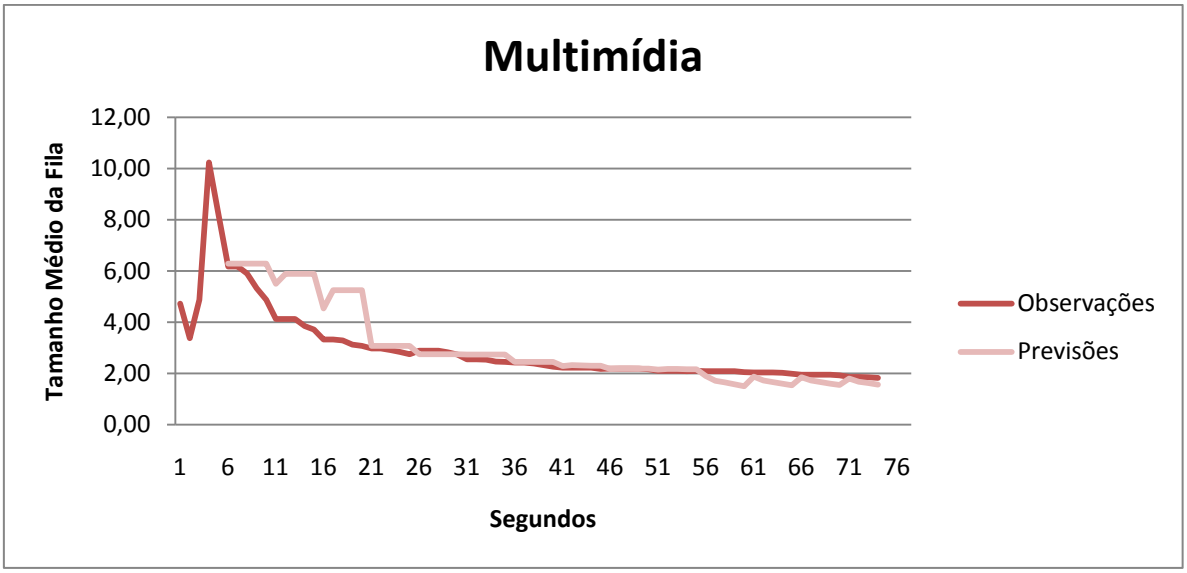


Figura 6.4: Previsão X Observações para o tráfego Multimídia.

Pode-se observar nas Figuras 6.2, 6.3 e 6.4 que no período de *warming up (1 a 6)* temos um acúmulo de requisições e não temos valores para as previsões. Ao longo do tempo, a partir das informações colhidas pelo método, o sistema converge e as previsões passam a estar muito próximas dos valores observados seguindo a tendência da série observada. Isto porque, com o passar do tempo, o *gateway* possui um histórico maior das observações.

A classe de aplicações sistemas críticos corresponde a aplicações que possuem *deadline* de 5 milissegundos. A classe voz interativa e multimídia possuem *deadlines* de 10 e 100 milissegundos respectivamente.

6.4 Modelo de Previsão Para Requisições de Esforço excelente, Melhor esforço e Segundo Plano

O foco principal deste estudo é analisar o comportamento das requisições que possuem algum *deadline*, em meio a ambientes de rede comuns, isto é, redes que possuem várias classes de serviço não só serviços de tempo real. Assim o módulo gerador foi configurado para enviar requisições de todas as classes de serviço aleatoriamente, para que seja possível

simular o comportamento do *gateway* em condições próximas as encontradas em redes contemporâneas.

Como as classes de serviço esforço excelente, melhor esforço e segundo plano não possuem *deadline*, sua priorização leva em consideração características das aplicações envolvidas e não os valores previstos. Deste modo a previsão para esse tipo de classe não se faz necessária. Porém, para a alocação de recursos é necessário observar as condições dessas filas nos mesmos instantes das observações feitas para as outras filas. Os gráficos a seguir ilustram as observações feitas para as filas de menor prioridade.

Figura 6.5: Observações para o tráfego esforço excelente.

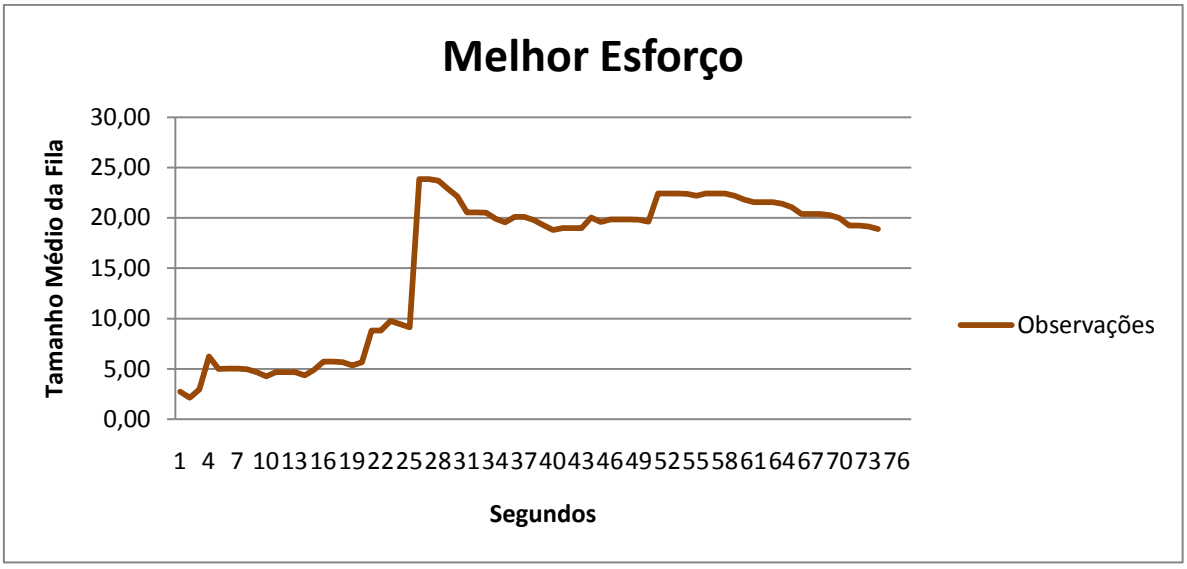


Figura 6.6: Observações para o tráfego melhor esforço.

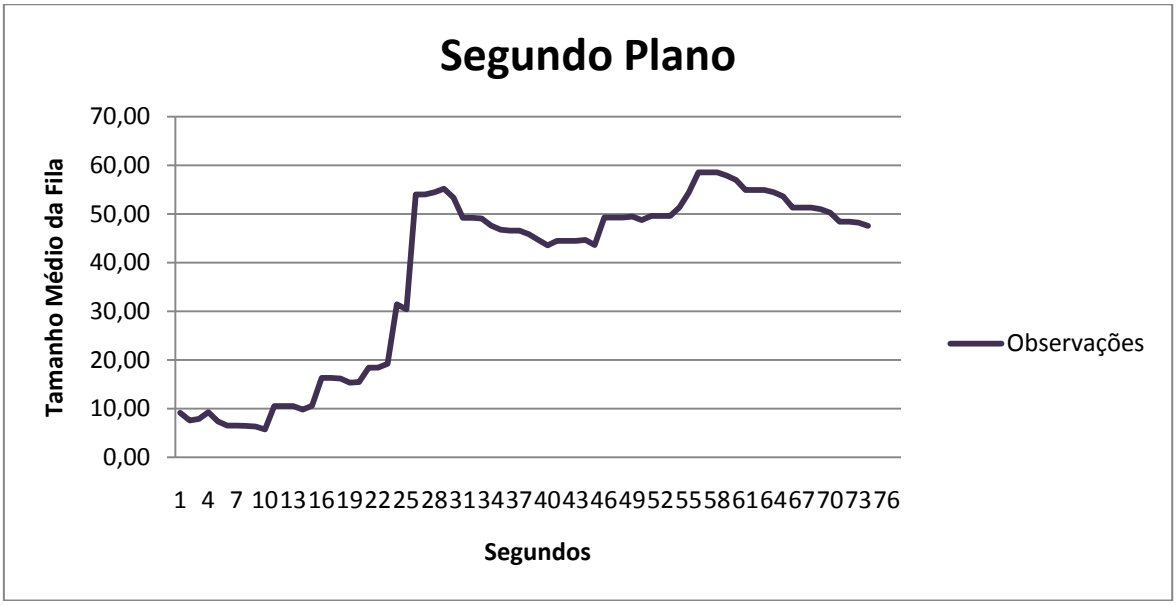


Figura 6.7: Observações para o tráfego segundo plano.

As Figuras 6.5, 6.6 e 6.7 ilustram as observações, do número médio de requisições na fila, feitas em períodos discretos de tempo. Para cada fila observa-se um comportamento independente, confirmando a eficiência do gerador.

6.5 Alocação uniforme

A alocação uniforme de recursos consiste em distribuir os processos destinados a retirar as requisições da fila uniformemente entre as filas propostas. Desta forma não existe diferenciação entre as classes de serviço. Esse tipo de abordagem é interessante quando o número de processos existentes é suficiente para atender todo o tráfego que passa pelo *gateway*.

Variando o número total de processos, estaremos dispondo de mais processos alocados para cada classe de aplicação, podendo comparar quantas requisições perderam seu *deadline* entre as diferentes abordagens. Para obter uma alocação uniforme entre as filas é necessário que o número total de processos disponíveis no sistema para atendimento das filas seja múltiplo do número de filas.

Para esse teste foram feitas duas abordagens diferentes. Na primeira foram alocados 30 (trinta) processos estaticamente, na segunda foram alocados 60 (sessenta) processos também estaticamente. Para cada abordagem os processos foram distribuídos igualmente entre as filas. Para simular uma situação de estresse mais facilmente foi adicionado atraso de 50 milissegundos em todos os processos. Para que não haja adulteração nos resultados, o mesmo atraso adicionado nos processos foi estendido para os *deadlines*.

A Figura 6.8, ilustra o comportamento das requisições, com restrições temporais, de acordo com a variação do número de processos. Para esse teste foram geradas aleatoriamente 40.000 mil requisições das diferentes classes de serviço.

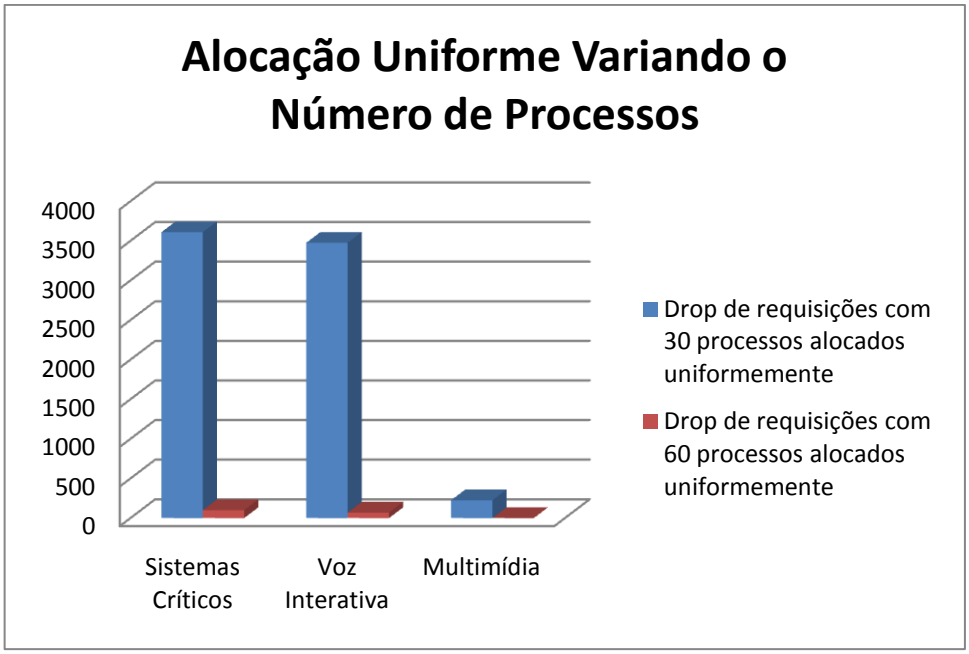


Figura 6.8: Alocação Uniforme Variando o Número de Processos.

A Figura 6.8 mostra que quando aumentamos o número de processos responsáveis pela retirada das requisições da fila, o número de requisições que perdem seu *deadline* cai consideravelmente.

6.6 Comparação entre Alocação Estática e Dinâmica de Processadores

A comparação entre alocação estática e dinâmica de processadores foi feita com base na quantidade de requisições que falham para cada uma das estratégias de alocação (i.e. estática e dinâmica). Entretanto para entender o comportamento geral do sistema, esse trabalho apresenta uma análise comparativa, entre as estratégias supracitadas, do comportamento das filas para cada classe de serviço, da alocação de processos ao longo do tempo e descarte das requisições com restrições temporais ao longo do tempo.

O procedimento de testes envolveu a exposição do protótipo à tráfegos idênticos de requisições para cada situação. Nos testes, os estados das variáveis e políticas dos módulos da

arquitetura são mantidos constantes, exceto, o número de processadores que varia de acordo com a estratégia de alocação utilizada.

Foram realizados testes comparando a alocação estática e dinâmica de 30 (trinta) processos, submetidos a um mesmo fluxo de requisições. Para esta abordagem, foram realizados testes com o envio de conjuntos de 40.000 (quarenta mil) requisições compostas pelas diferentes classes de serviço geradas aleatoriamente. A Tabela 6.2 demonstra a distribuição das requisições por classe de serviço.

Tabela 6.2 Distribuição de requisições por classe de serviço.

Descrição da classe	Número de requisições
Sistemas críticos	6.528
Voz interativa	6.618
Multimídia	6.670
Esforço excelente	6.928
Melhor esforço	6.627
Segundo Plano	6.629

Cada teste analisou a quantidade de requisições canceladas, ou seja, cujos *deadlines* expiraram por espera de atendimento nas filas.

Para determinar de modo mais confiável qual a influência do tempo de espera por atendimento nas filas do *gateway*, no sucesso ou falha de uma requisição, foi adicionado atrasos aos processos que atendem as filas do *gateway*. Esses atrasos são iguais para todos os processos e aumentam a unidade temporal utilizada para medida de *deadline*.

Para não penalizar a execução das requisições com o atraso adicionado aos processos, o mesmo também foi propagado para o *deadline* das requisições.

Na realização dos testes fixou-se o atraso dos processadores em 50 (cinqüenta) milissegundos, e a política de inserção utilizada para as filas foi *First In, First Out* (FIFO).

6.6.1 Comparação do Comportamento das Filas

Observar o comportamento das filas ajuda a entender por que a alocação dinâmica diminui o número de perda de *deadline* em relação a alocação estática. As filas apresentam comportamentos semelhantes nos instantes iniciais, isso porque o *gateway* está no período de *warming up*. Após o *warming up* as filas com tráfego com requisitos temporais têm uma queda brusca.

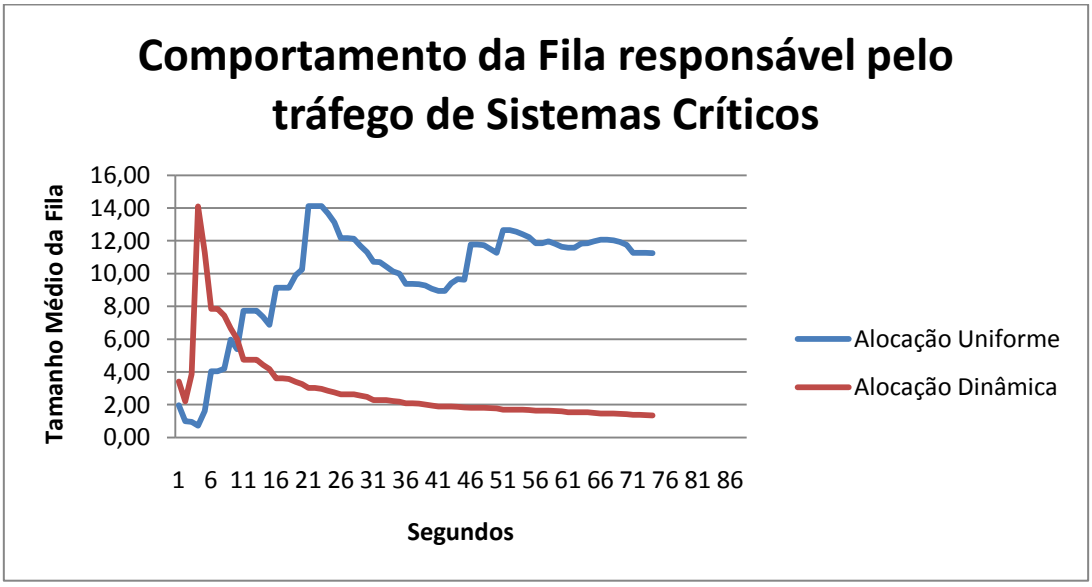


Figura 6.9: Comparação entre o comportamento da fila responsável pelo tráfego de Sistemas Críticos para alocação uniforme e dinâmica.

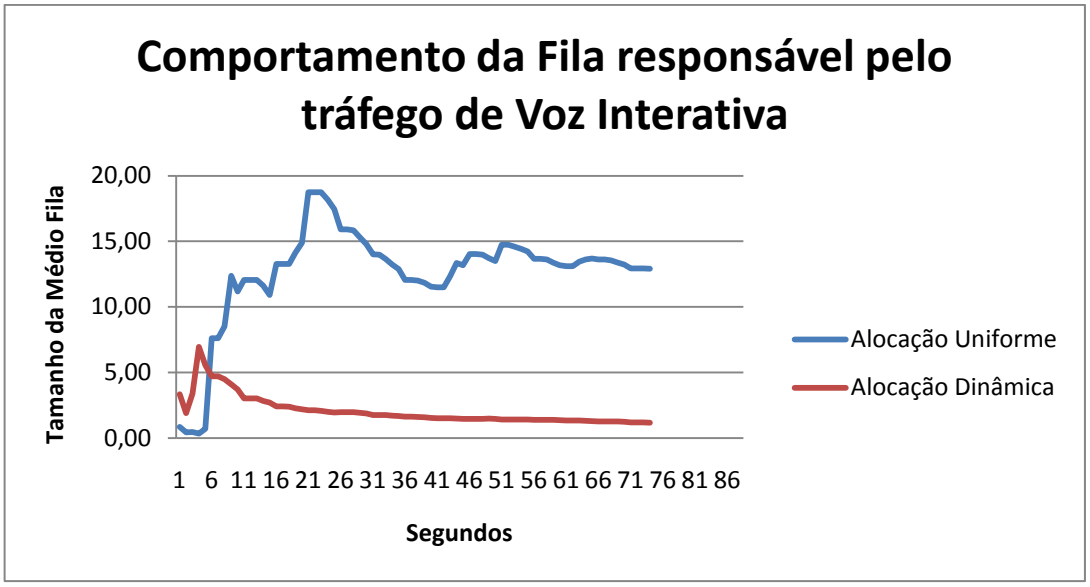


Figura 6.10: Comparação entre o comportamento da fila responsável pelo tráfego de Voz Interativa para alocação uniforme e dinâmica.

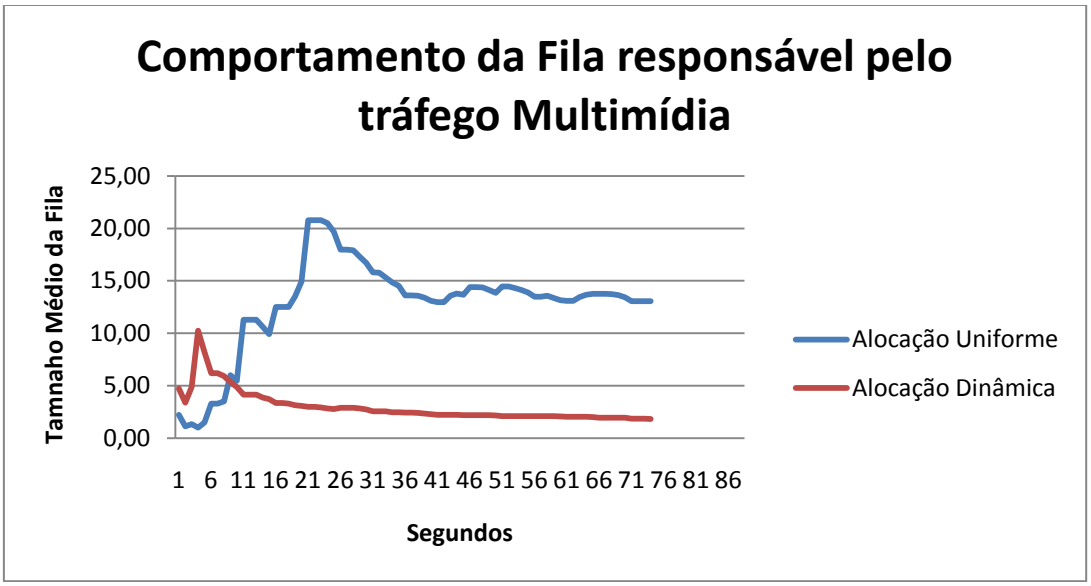


Figura 6.11: Comparação entre o comportamento da fila responsável pelo tráfego Multimídia para alocação uniforme e dinâmica.

Para as filas com tráfego sem restrições temporais podemos ver um comportamento inverso do tamanho das filas. Na alocação dinâmica o tamanho da fila é superior a alocação estática. Este comportamento já é esperado visto que os processos que estavam dedicados a atender as filas com menor prioridade passaram a atender as filas mais prioritárias.

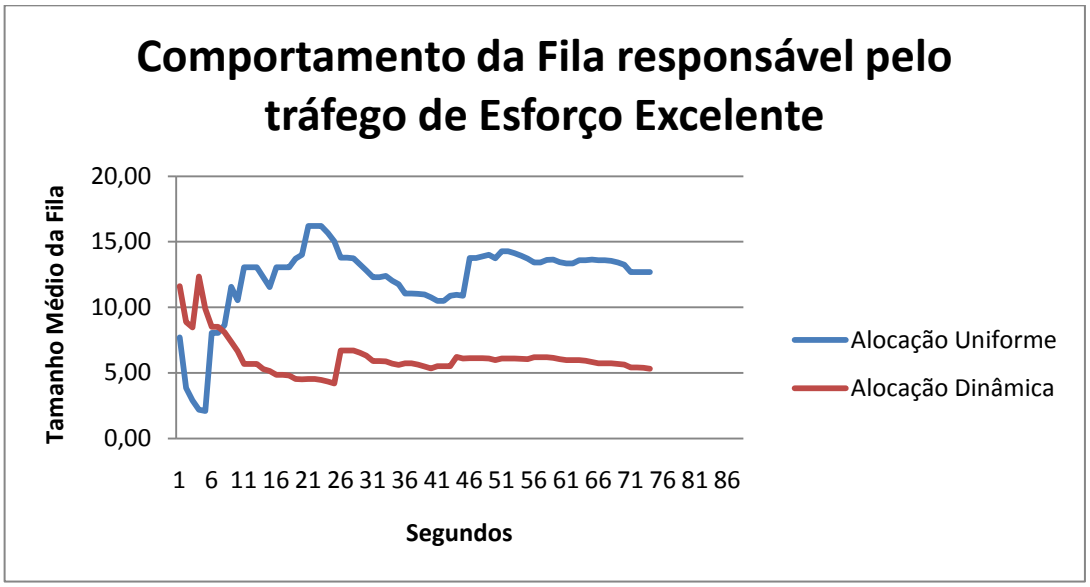


Figura 6.12: Comparação entre o comportamento da fila responsável pelo tráfego de Esforço Excelente para alocação uniforme e dinâmica.

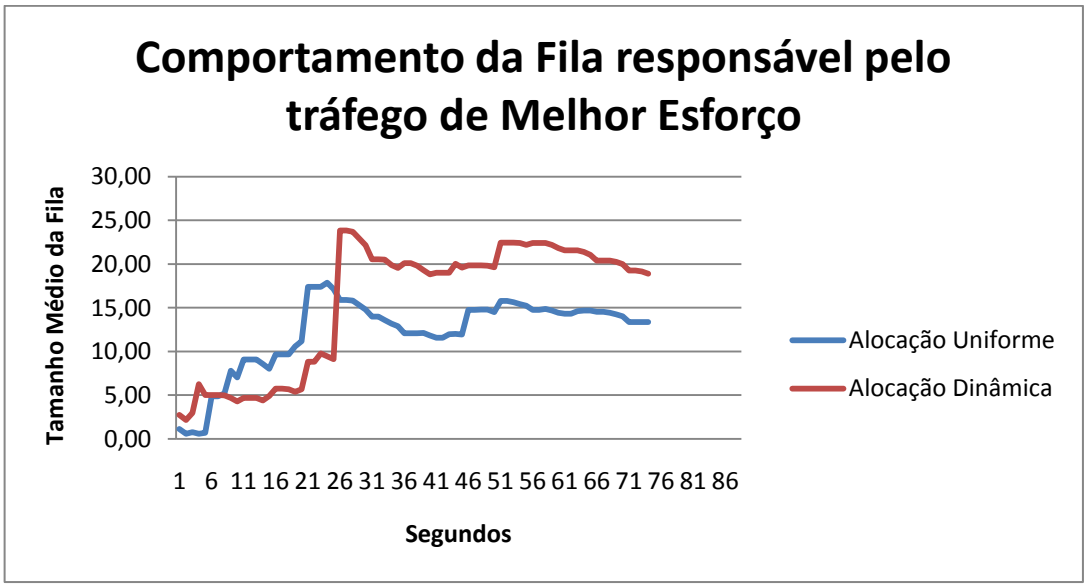


Figura 6.13: Comparação entre o comportamento da fila responsável pelo tráfego de Melhor Esforço para alocação uniforme e dinâmica.

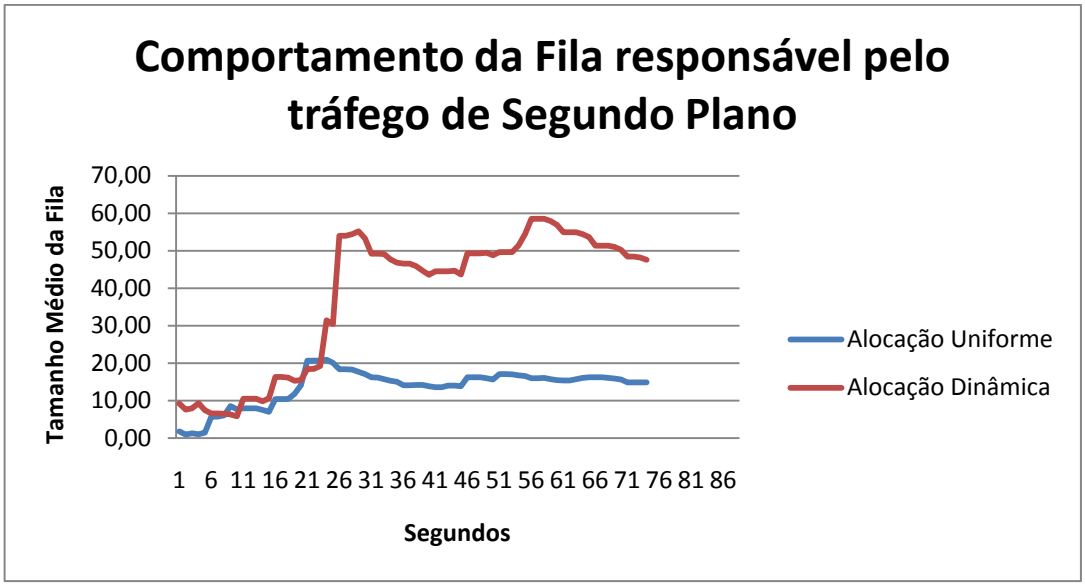


Figura 6.14: Comparação entre o comportamento da fila responsável pelo tráfego de Segundo Plano para alocação uniforme e dinâmica.

6.6.2 Perda de *Deadline*

Analisar a quantidade de requisições que perderam seu *deadline* nos mostra a eficiência do *gateway*, tendo em vista que o objetivo do *gateway* é diminuir o número de requisições que foram descartadas por terem perdido seu *deadline*.

A Figura 6.15 mostra uma comparação entre a utilização da alocação uniforme com a alocação dinâmica.

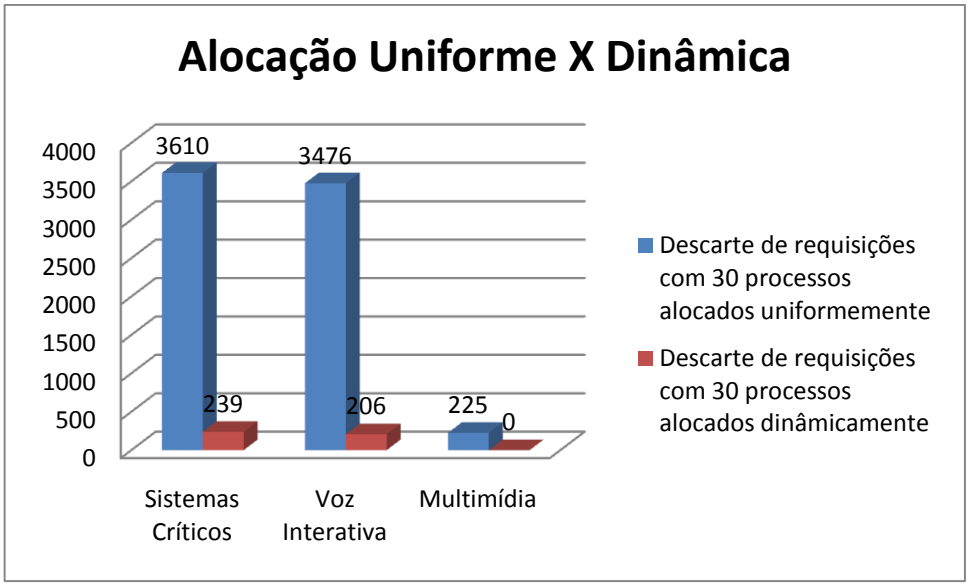


Figura 6.15: Descarte de requisições.

Pode-se observar uma queda significativa no número de requisições que perderam seu *deadline*. Para a classe de sistemas críticos houve uma redução de 93,38% de descarte de requisições. Para a classe de voz interativa a queda no número de descarte foi de 94,08% e para a classe de aplicações multimídia a queda foi de 100%. Assim fica evidente a vantagem de se utilizar a alocação dinâmica proposta para as classes de serviços utilizadas nesse trabalho.

6.6.3 Considerando o Período de *Warming Up*

No *warming up* o *gateway* utiliza a distribuição uniforme nos instantes iniciais de execução para coletar dados e efetuar as previsões. Para saber efetivamente quantas requisições perderam seu *deadline* na alocação dinâmica, é interessante comparar o número de requisições que perderam o *deadline* no período de *warming up* com o total de requisições que perderam seu *deadline* ao longo da simulação.

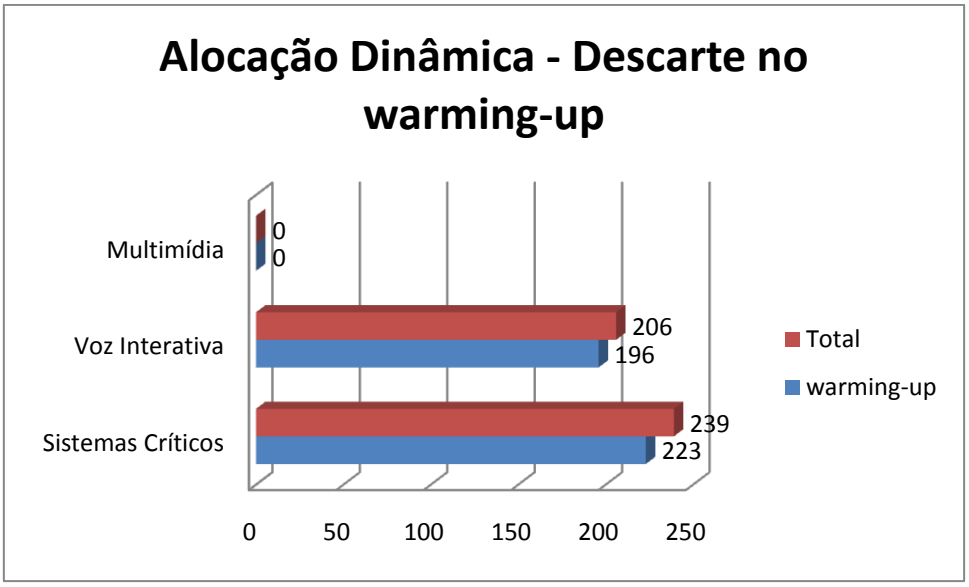


Figura 6.16: *Warming up* descarte de requisições.

Pode-se observar na Figura 6.16 que na classe de aplicações multimídia, que possui *deadline* maior que as demais, nenhuma requisição perdeu seu *deadline* tanto no período de *warming up* quanto na alocação dinâmica efetiva. Para a classe voz interativa o período de *warminig up* representa aproximadamente 95,15% das perdas de *deadline* em relação ao total de requisições perdidas ao longo de toda a simulação. Para a classe sistemas críticos o período de *warminig up* representa aproximadamente 93,30% do total de requisições que perderam seu *deadline*.

6.6.4 Descarte ao Longo do Tempo

Nesta seção é apresentada uma comparação da alocação uniforme com a alocação dinâmica com o intuito de ilustrar o comportamento do *gateway* em relação a perda de *deadline* em períodos regulares. A cada 5 (cinco) segundos o *gateway* grava no banco de

dados as estatísticas de cada fila. Fazendo a diferença entre cada período, temos o número de requisições que perderam seu *deadline* por período.

As Figuras 6.17, 6.18 e 6.19 ilustram o número de descarte por *deadline* ao longo do tempo para as filas que possuem requisições com restrições temporais comparando a alocação uniforme com a alocação dinâmica.

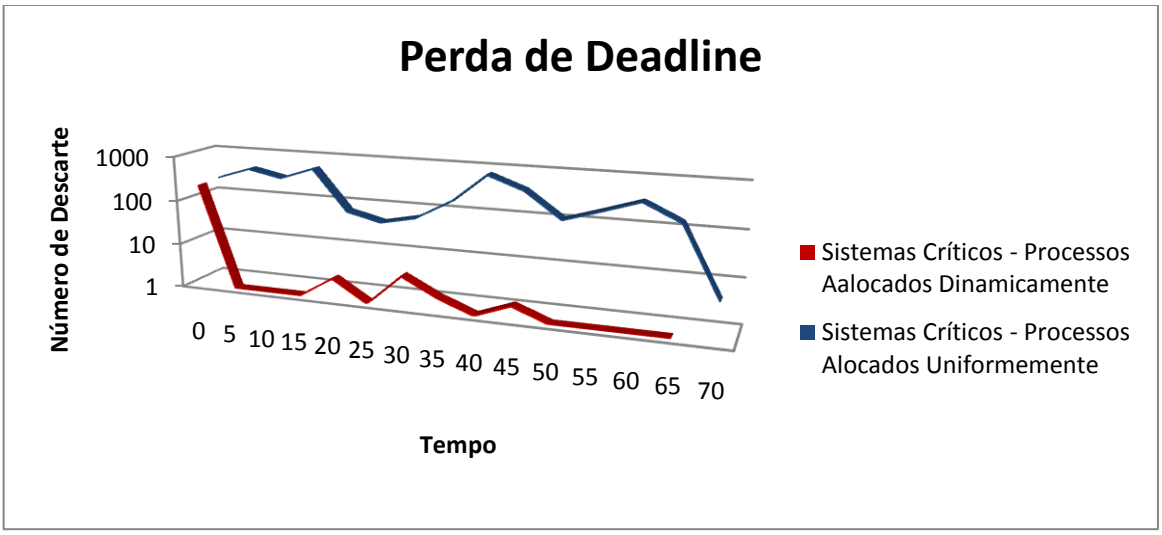


Figura 6.17: Descarte ao logo do tempo para Sistemas Críticos.

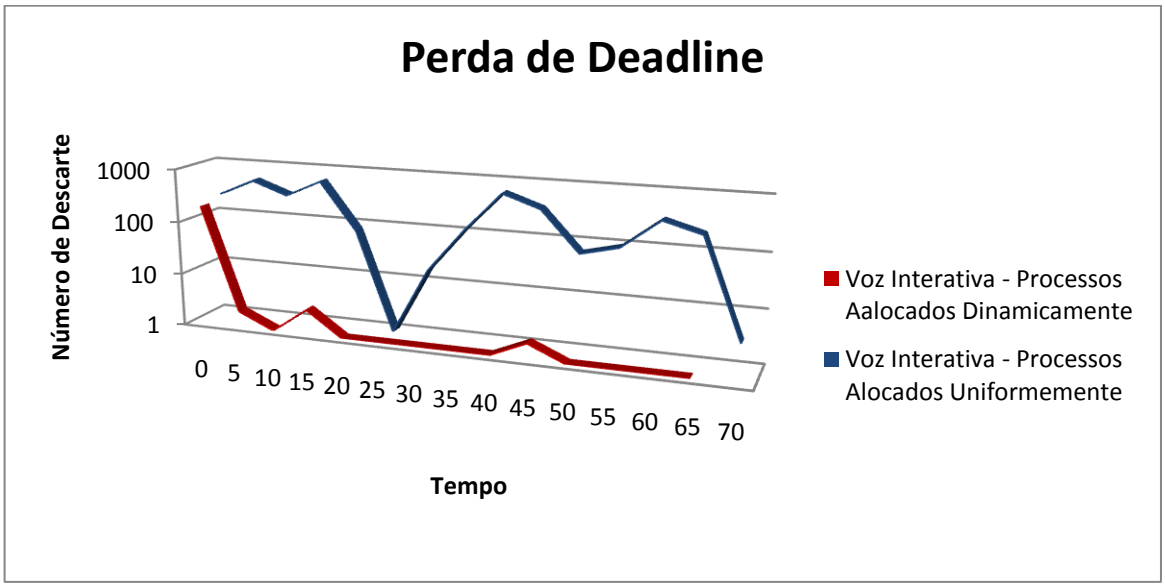


Figura 6.18:Descarte ao logo do tempo para Voz Interativa.

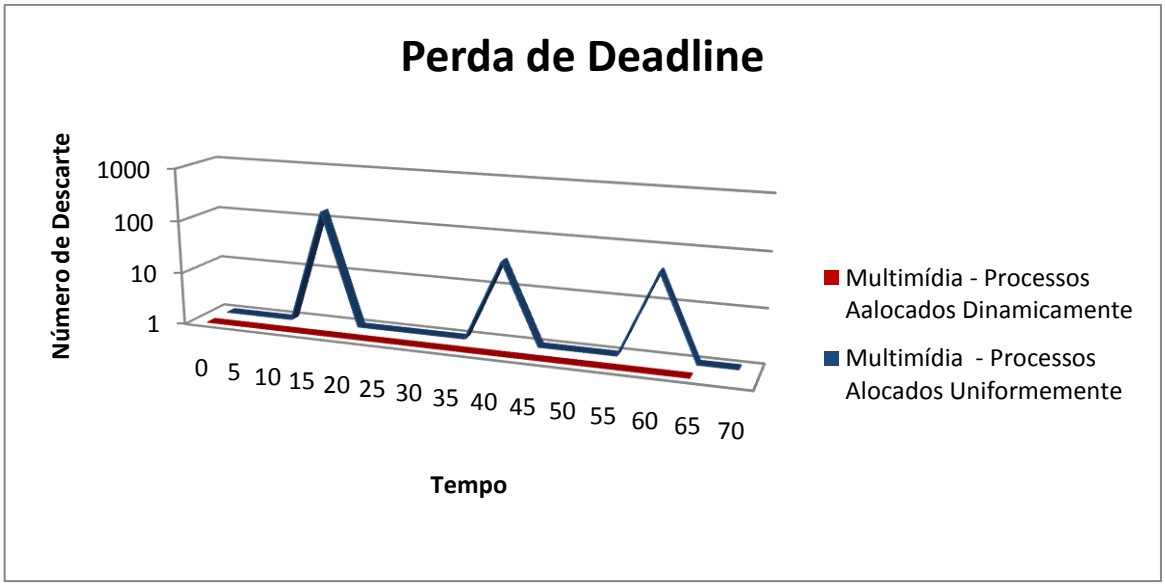


Figura 6.19: Descarte ao logo do tempo para Multimídia.

7 CONCLUSÃO

Os testes realizados sobre o protótipo validam a arquitetura proposta. Os resultados mostram que a arquitetura consegue atingir seu objetivo de privilegiar as requisições de prioridades mais altas, principalmente a classe de serviço com requisitos temporais mais rígidos.

Mesmo com o período inicial de *Warming up* e com variações no tráfego das diferentes classes aplicações vigentes na rede, os modelos de previsão baseados no método de *Box-Jenkins* conseguem realizar boas previsões sobre o volume e características do tráfego para instantes brevemente futuros seguindo a tendência da série real. Dessa forma, a arquitetura pode se ajustar antecipadamente, com bastante precisão, aos estados futuros do ambiente de rede.

As previsões permitem alocar dinamicamente o número de processadores próximo do ideal para atender a demanda de requisições com requisitos temporais. A alocação dinâmica de processadores visa a melhor utilização dos recursos disponíveis no *gateway*. O estudo de caso mostrou que a alocação estática de processadores não atende de forma satisfatória as filas de maior prioridade tendo um número maior de requisições que perderam seu *deadline*.

Já na alocação dinâmica temos um comportamento adequado para as filas de maior prioridade, resultando em uma diminuição significativa no número de requisições que perderam seu *deadline*. Conseqüentemente podemos observar um crescimento nas filas de menor prioridade, o que prova que o *gateway* está cumprindo com sua proposta de priorizar o tráfego de maior prioridade.

A alocação dinâmica de processos com base nas previsões, otimiza a utilização dos recursos e busca manter a ocupação das filas sempre a menor possível. A alocação dinâmica de processos garante também que sempre a maior parte dos processos estão ocupados, fazendo sempre boa utilização dos recursos alocados.

A estratégia de *Warming up* mostrou-se muito boa visto que inicialmente não é necessário utilizar nenhum outro tipo de analisador de tráfego para descobrir o comportamento da rede e obter dados para a previsão inicial. Assim a arquitetura ora proposta mostrou trabalhar de forma autônoma sendo altamente adaptável a contexto de rede em que é inserida.

Os resultados também mostram que mesmo com todas as otimizações implementadas no protótipo utilizado para simulação, parte das requisições ainda é perdida. Isto se deve ao fato dos requisitos de tempo serem extremamente rígidos para os tráfegos testados e ao período de cada realocação de processos.

7.2 Trabalhos Futuros

Os trabalhos futuros envolvem um estudo do Gerenciador de Processos para verificar a melhor política de alocação de recursos, visto que uma pequena variação no tempo entre o cálculo das alocações e o intervalo de coleta dos dados pode alterar o resultado. Assim a política de alocação bem como o intervalo entre as coletas também poderiam ser feitos dinamicamente otimizando ainda mais o desempenho do *gateway*.

Outro aspecto é integrar o sistema de previsão ao Módulo Estatístico para que o sistema de previsão seja feito todo no *gateway* sem a dependência de programas de terceiros.

Ainda podemos analisar o comportamento do *gateway* para outras classes de aplicações com comportamentos diferentes da aplicadas neste trabalho.

Outros trabalhos incluem a adaptação da arquitetura para novas perspectiva aplicadas a capacidade de suportar QoS pela infra-estrutura de rede sem a necessidade de alteração nos protocolos de camadas inferiores. Uma adaptação para essa abordagem é aprimorar o

Gerenciador de Conexão para que ele possa identificar os protocolos e classificá-los nas diversas classes de serviços da rede.

8 REFERÊNCIAS

ABDEL-AAL, R. E.; AL-GARNI, Z. **Forecasting monthly electric energy consumption in eastern Saudi Arabia using univariate time-series analysis.** Energy, n. 11, v. 22, p. 1059-1069, 1997.

AUDSLEY, N. C.; BURNS A.; RICHARDSON M.; TINDELL, K.; WELLINGS, A. J. **Applying New Scheduling Theory to Static Priority Pre-Emptive Scheduling.** Software Engineering Journal, 8(5):284 – 292, 1993.

BENGTSSON, J.; GRIFFIOEN, W. O. D.; KRISTOFFERSEN, K. J.; LARSEN, K. G.; LARSSON, F.; PETTERSSON, P.; YI, W. **Verification of an Audio Protocol with Bus Collision Using UPPAAL.** In 8th International Conference on Computer Aided Verification CAV, volume 1102 of Lecture Notes in Computer Science, pages 244–256, July/August 1996.

BEIZER, B. **The Pentium bug, an industry watershed.** Testing Techniques Newsletter, On-Line Edition, Setembro, 1995.

BERTSEKAS, D.; GALLAGER, R. **Data networks.** 2Ed. Englewood Cliffs: Prentice-Hall International, 1992.

BORGES, R. B.; OLIVEIRA, R. S. **Análise temporal da implementação da fila de aptos como lista ordenada e lista desordenada.** 3o WSO - Workshop de Sistemas Operacionais Campo Grande - MS, 17 de julho de 2006.

BOX, G. E. P.; JENKINS, G. M. **Time series analysis.** Holden Day, 1976.

CARRO, L.; WAGNER, F. R. **Sistemas Computacionais Embarcados.** In JAI'03 - XXII Jornadas de Atualização em Informática, Campinas, Brasil, August 2003.

CRUZ, G. M.; LIMA, G. **Simulador de Escalonamento para Sistemas de Tempo Real.** In: IV WTICG - ERBASE 2006, 2006, Aracajú. IV WTICG - trabalho de conclusão de curso, 2006. p. 1-11.

FARINES, J. M.; FRAGA, J. S.; OLIVEIRA, R. S. **Sistemas de Tempo Real**. 12ª Escola de Computação, IME-USP, São Paulo-SP, julho de 2000.

FRANCIA, G. A. **Embedded Systems Programming.** The Journal of Computing in Small Colleges, v. 17, n. 2, p. 204-210, dez. 2001. Proceedings of the 15th Annual CCSC Southeastern Conference.

FRAGA, J.; FARINES, J. M.; FURTADO, O.; SIQUEIRA, F. **A Programming Model for Real-Time Applications in Open Distributed Systems.** In: FTDCS'95 Proceedings, IEEE Pub., Korea. 1995.

FINE, S.; UR, S.; ZIV, A. **Probabilistic Regression Suites for Functional Verification.** In Proceeding of the Design Automation Conference (DAC), 2004.

GANZ, A.; WONGTHAVARAWAT, K.; PHONPHOEM, A. **Q-soft: software framework for qos support in home networks.** Computer Networks, v. 42, p. 7–22, 2003.

GROSS, D.; HARRIS, C.M. **Fundamentals of Queueing Theory**, John Wiley & Sons, 1985.

GONÇALVES, R. P.; OLIVEIRA, R. S.; MONTEZ, C. **Design Patterns for the Adaptive Scheduling of Real-Time Tasks with Multiple Versions** in RTSJ. XXV International Conference of the Chilean Computer Science Society - SCCC'2005 Valdivia - Chile, november 2005.

HENNESSY, J. L.; PATTERSON, D. A. **Arquitetura de Computadores: Uma abordagem Quantitativa**, volume 1. Ed Campus, 3a edition, 2003.

HWANG, W.; TSENG, P.; **"QoS-Aware Residential Gateway with Bandwidth Management"**, iih-msp, pp. 173-176, 2006 International Conference on Intelligent Information Hiding and Multimedia, 2006.

IEEE 802.1p Integrated Service Mappings on IEEE 802 Networks http://www.ietf.org/internet-drafts/draft-ietf-issll-is802-svc-mapping-04.txt

LEE, J.; **"Design of a Communication System Capable of Supporting Real-Time RPC"** ,iceccs, p. 99, 1996.

LEI, B.; ANANDA, A.; TECK, T. S. **"QoS-aware Residential Gateway"** in Proceedings of the 27th Annual IEEE Conference on Local Computer Networks (LCN'02), 2002.

LEHOCZKY, J. P.; SHA, L.; STROSNIDER, J.K. **Enhaced Aperiodic Responsiveness in Hard Real_time Environments.** Proceedings of IEEE Real-Time Systems Symposium, San Jose, CA, pp. 261-270, 1987.

LIU, J. W. S. **Real-Time Systems.** Prentice-Hall, 2000.

MACÊDO, R. J. A.; LIMA G. M.; BARRETO, L. P.; ANDRADE, A. M. S.; BARBOZA F. J. R.; SÁ A.; ALBUQUERQUE, R.; ANDRADE, S. **Tratando a previsibilidade em sistemas de tempo-real distribuídos: Especificação. Linguagens,** ***Middleware*** **e Mecanismos Básicos.** In Capítulo 3 do Livro texto para o mini-curso a ser apresentado no 22o. Simpósio Brasileiro de Redes de Computadores, SBRC'2004.pp. 105-163, ISBN: 85-88442-82-5 Gramado, RS, May 2004.

MAISTER, D. H. **Note on the management of queues.** Harvard Business Review, p. 1–14, 1995.

MATSCHULAT, D. **Provisão de Qualidade de Serviço em Escalonadores para Sistemas Operacionais Embarcados de Tempo-Real.** Dissertação de Mestrado, PUCRS, Porto Alegre, 2007.

MORETTIN, P. A.; TOLOI, C. M. **Previsão de Séries Temporais**, 2a ed. São Paulo: Editora Atual, 1987.

MORETTIN, P. A.; TOLOI, C. M. Análise de séries temporais, ISBN: 8521203896**,** Editora Edgard Blücher, 2004.

NEWSGENERATION http://www.rnp.br/newsgen/9911/qos.html Acesso em 22/04/2008

NOBILE, P. N. **"Uma Arquitetura com Qualidade de Serviço para Chamadas Remotas de procedimentos de Tempo Real"**, dissertação apresentada ao Programa de Pós-Graduação em Ciência da computação da UFSCar.

NOBILE, P. N.; LOPES, R. R. F.; MORON, C. E.; TREVELIN, L. C. **"RPC-RT Proxy: Um proxy para RPCs de tempo real utilizando o método de previsão Box-Jenkins"**, XXV Simpósio Brasileiro de Redes de Computadores, 2007, Belém.

NOBILE, P. N.; LOPES, R. R. F.; MORON, C. E.; TREVELIN, L. C. **"QoS Proxy Architecture for Real Time RPC with Traffic Prediction"**, The 11-th IEEE International Symposium on Distributed Simulation na Real Time Applications", October 22 – 24, 2007, Chania, Crete Island, Greece.

OLIVEIRA, R. S. **Aspectos Construtivos dos Sistemas Operacionais de Tempo Real.** WTR'2003 - 5o Workshop de Tempo Real Natal - RN, 22 de maio de 2003.

OLIVEIRA, R. S.; CARISSIMI, A. S.; TOSCANI, S. S. **Organização de sistemas operacionais convencionais e de tempo real.** XXI JAI - Jornada de Atualização em Informática Porto Alegre - RS, 2002, v. 1, p. 327-378.

PEIXOTO, M. L. M.; TOTT, R.; NERY, M.; MONACO. F. J. **Arquitetura de Escalonamento Ortogonal de Tempo-Real para garantias de QoS em ServidoresWeb.** SBC 2008, Workshop em Desenpenho de Sistemas Computacionais e de Computação, Belém do Para, 2008.

PLENTZ, P.; MONTEZ, C.; OLIVEIRA, R. S.; FRAGA, J. S. **Programação Baseada em Threads Distribuídas nas Especificações RT-CORBA 2.0 e Distributed RTSJ.** WTR'2003 - 5º Workshop de Tempo Real Natal - RN, 22 de maio de 2003.

TATIBANA, C. Y.; MONTEZ C.; OLIVEIRA R. S. **Soft Real-Time Task Response Time Prediction in Dynamic Embedded Systems**, SEUS'2007 - The 5th IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems Santorini Island, Greece, 7-8 May 2007.

TATIBANA, C. Y.; MONTEZ, C.; OLIVEIRA, R. S. **Real-time Dynamic Guarantee in Component-based Middleware.** ISORC'2007 - The 10th IEEE International Symposium on Object/component/service-oriented Real-Time Distributed Computing Santorini Island, Greece, 7-9 May 2007.

RECH, L. O.; MONTEZ, C.; OLIVEIRA, R. S. **A New Model for the Itinerary Definition of Real-Time Imprecise Mobile Agents.** IRI'2006 -IEEE International Conference on Information Reuse and Integration Waikoloa, Hawaii, USA, 16-18 September 2006.

RECH, L.; MONTEZ, C.; OLIVEIRA, R. S. **Determinação Dinâmica do Itinerário de Agentes Móveis Imprecisos com Deadline Firme.** WTR'2006 - 8o Workshop de Tempo Real Curitiba - PR, 02 de junho de 2006.

ROCHA, F. R.; OLIVEIRA, R. S. **Escalonamento baseado em Intervalo de Tempo.** SBAI'2007 - Simpósio Brasileiro de Automação Inteligente Florianópolis, SC, 8-11 de Outubro de 2007.

ROCHA, F. R.; OLIVEIRA, R. S. **Time-Interval Scheduling and its Applications to Real-Time Systems.** The 27th IEEE Real-Time Systems Symposium, work-in-progress session Rio de Janeiro - RJ, Brazil, 5-8 December 2006.

ROCHA, F.; OLIVEIRA, R. S. **Modelo de Tarefas Baseado em Instante Ideal.** WTR'2005 - 7o Workshop de Tempo Real Fortaleza - CE, 13 de maio de 2005.

ROCHA, F.; OLIVEIRA, R. S. **Ferramenta para Concepção de Sistemas de Arquivos de Tempo Real Embutidos.** WSO'2004 - 1o Workshop de Sistemas Operacionais Salvador - BA, 5-6 de agosto de 2004.

TATIBANA, C. Y.; OLIVEIRA, R. S.; MONTEZ, C. **Um Estudo das Propostas de Modelos de Componentes para Sistemas de Tempo Real.** WTR'2004 - 6o Workshop de Tempo Real Gramado - RS, 14 de maio de 2004.

SILVA, K. R. G.; MELCHER, E. U. K.; ARAUJO, G. **An Automaic Testbench Generation Tool for a SystemC Functional Verification Methodology.** In: SBCCI2004 – 17th Symposium on Integrated Circuits and System Design, Porto de Galinhas – PE, 2004.

SURI, R.; SANDERS, J.L.; KAMATH, M. **"Performance evaluation of production network"** Graves, S.C.; Kan, A.H.G.R.; Zipkin, P.H.; (editores) Handbooks in Operational Research and Management Science Vol 4, 199-286, Elsevier Science Publishers B.V.,1993.

SCHILL, A.; KUMMEL, S.; HUTSCHENREUTHER, T. **"QoS Support for Advanced RPC Interactions"**, mmnet, p. 0245, 1997.

SCHWARTZ, M. **Computer-communication network design and analysis.** New York: Prentice-Hall, 1977.

SCHWARTZ, M. **Telecommunication networks: protocols, modelling and analysis.** New York: Addison-Wesley, 1987.

SHA, L.; RAJKUMAR, R.; LEHOCZKY, J. P. **Priority Inheritance Protocols: Na approach to Real-Time Synchronization.** IEEE Transactins on Computers, Vol. 39, No. 9, pp. 1175-1185, september 1990.

SPRUNT, B.; SHA, L.; LEHOCZKY, J. **Aperiodic Tasck Schesuling for Hard Real-Time Systems.** The Journal Of Real-Time Systems, Vol 1, pp. 27-60,1989.

TANENBAUM, A. S.; STEEN, M. V. **Distributed systems: Principles and paradgmas.** Englewood Cliffs: Prentice Hall, 2002.

TANENBAUM A. S. **Redes de computadores.** 4ª edição, Editora Campus, 2003.

VILLELA, R. T. N. **Gerador de interfaces gráficas com suporte à chamada remota de procedimento de tempo real.** Dissertação de Mestrado, UFSCar, São Carlos, 2001.

WANG, Q.; YE, Q.; CHENG, L. **"An Inter-application and Inter-client Priority-based QoS Proxy Architecture for Heterogeneous Networks"**, iscc, pp. 819-824, 10th IEEE Symposium on Computers and Communications (ISCC'05), 2005.

WEISER, M.; **"The computer for 21st century"**. Scientific American, p. 94-104, 1991.

WERNER, L.; RIBEIRO, J. L. D. **Previsão de demanda: Uma aplicação dos modelos de box-jenkins na área de assistência técnica de computadores pessoais.** Gestão & Produção, v. 10, n. 1, p. 47–67, 2003.

WIKIPEDIA http://pt.wikipedia.org/wiki/Televis%C3%A3o_digital Acesso em 23/07/2009