TARCÍSIO PAULO CORLASSOLI

UMA POLÍTICA DE ESCALONAMENTO PARA SERVIDORES WEB BASEADA NA VELOCIDADE DA CONEXÃO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática, Curso de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Prof.ª Cristina Duarte Murta

CURITIBA
2001

UFPR
Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática do aluno ***Tarcísio Paulo Corlassoli***, avaliamos o trabalho intitulado ***"Política de Escalonamento para Servidores Web baseada na Velocidade da Conexão"***, cuja defesa foi realizada no dia 20 de setembro de 2001, às quatorze horas, no anfiteatro B, do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 20 de setembro de 2001.

Profª. Dra. Cristina Duarte Murta
DINF/UFPR - Orientadora

Prof. Dr. Virgílio Augusto Fernandes Almeida
Membro Externo - DCC/UFMG

Prof. Dr. Elias Procópio Duarte Jr.
DINF/UFPR

TARCÍSIO PAULO CORLASSOLI

UMA POLÍTICA DE ESCALONAMENTO PARA SERVIDORES WEB BASEADA NA VELOCIDADE DA CONEXÃO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática, Curso de Pós-graduação em Informática da Universidade Federal do Paraná.

Orientadora: Profª. Cristina Duarte Murta

CURITIBA
2001

AGRADECIMENTOS

A Deus, por dar-me tranqüilidade nos momentos difíceis para superá-los.

À professora Cristina, pelo acompanhamento, orientação e, principalmente, pelo apoio durante o desenvolvimento deste trabalho.

À minha família, fonte de inspiração e motivação.

Aos meus colegas de trabalho, pelo incentivo e colaboração.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Esta dissertação propõe uma nova política de escalonamento para o processamento de requisições HTTP estáticas em servidores Web. Esta nova política chama-se FCF (*Fastest Connection First*). A política proposta atribui prioridades às requisições HTTP baseando-se no tamanho do arquivo solicitado e na velocidade da conexão com o usuário. As requisições para arquivos menores feitas através de conexões mais rápidas recebem maior prioridade. O que motivou a proposição desta política de escalonamento foi a distribuição dos tamanhos de arquivo transferidos na Web, a diversidade de condições de conectividade observadas na Internet e a possibilidade de saber com antecedência o tamanho do processo para atender a requisições estáticas. O objetivo da política FCF é otimizar a interação entre servidor Web e Internet visando um menor tempo final de resposta. A nova política foi comparada através de simulação com as políticas de uso corrente em servidores Web e também com a política SRPT *(Shortest Remaining Processing Time)*. Os resultados apresentam evidências de que as diferenças de conectividade observadas na Internet afetam o desempenho do servidor, e que essa informação pode ser utilizada para melhorar significativamente o desempenho do sistema. Além disso, comprovou-se que a distribuição dos tamanhos de arquivo da Web evita que ocorra *starvation* de processos grandes quando aplica-se políticas de escalonamento que priorizem processos pequenos.

ABSTRACT

This dissertation proposes a new scheduling policy for the processing of static HTTP requisitions in Web servers. This policy, called FCF (Fastest Connection First), gives priority to HTTP requests based on the size of the requested file and on the speed of the user's connection. The requests for smaller files through faster connections receive the highest priorities. The motivation of this proposal is the distribution of the file sizes transferred in the Web, the diversity of the effective bandwidth of the user's connection observed in the Internet and the possibility of knowing the size of the process in advance. The new policy was compared through simulation with the policies standard in Web servers and also with the policy SRPT (Shortest Remaining Processing Time). The results show evidences that the different levels of connectivity in the Internet affect the performance of the Web server, and that this information can be used to improve the performance of the system significantly. We showed that the distribution of the Web files sizes avoids starvation of big processes if a size based scheduling policy is applied.

1. INTRODUÇÃO

Servidores Web podem ter que tratar um grande número de requisições simultaneamente, oriundas de um imenso e heterogêneo universo de usuários. A diversidade de usuários diz respeito não apenas à cultura, idioma e localização geográfica, mas também a plataformas de hardware, software e, principalmente, a condições de conectividade. Pesquisa realizada pelo *Graphics, Visualization, & Usability Center* do *Georgia Institute of Technology* [10] mostrou que, em 1998, as condições de conectividade variavam até seiscentas vezes, por exemplo, de um modem de 14,4 Kbps para uma conexão de 10 Mbps. Além da largura de banda, fatores dinâmicos como a quantidade de tráfego no *link*, o número de *hops* no caminho de uma conexão TCP, o atraso, a taxa de transmissão dos *links* intermediários e características das diversas implementações do protocolo TCP contribuem para a variabilidade das condições de conectividade observadas na Internet. Alguns trabalhos mostram que as variações na Internet exercem grande influência no desempenho dos servidores Web [05, 32, 34]. No entanto, apesar da observação desta variabilidade e de sua influência sobre o desempenho destes servidores, não são encontrados na literatura trabalhos sobre políticas de escalonamento para servidores Web que levem em consideração a interação do servidor com a Internet.

Além da variabilidade nas condições de conectividade, outras duas características do ambiente Web podem ser exploradas no projeto de novas políticas de escalonamento. Estas características são a possibilidade de conhecer antecipadamente o tamanho dos processos para tratamento de requisições HTTP estáticas e a variabilidade do tamanho dos arquivos requisitados. Para requisições HTTP estáticas, que são a maioria das requisições Web [03], o tempo de processamento pode ser considerado proporcional à quantidade de bytes servidos [04, 09, 17, 18, 19]. Desta forma, é possível prever o tamanho do processo antes do atendimento da requisição.

A teoria tradicional de escalonamento de processos mostra que quando o custo dos processos é conhecido com antecedência, uma política que dê prioridade aos processos menores diminui o tempo médio de resposta [25, 48]. Em sistemas nos quais a política de escalonamento não considera o tamanho do processo para definir a prioridade, as requisições pequenas esperam pelo atendimento de requisições muito grandes, aumentando assim o tempo médio de resposta. O principal argumento contra o

uso desta política é a possibilidade de ocorrência da chamada *starvation* dos processos grandes. Esta é uma preocupação legítima para várias distribuições dos tamanhos das tarefas mas não para o caso das distribuições de cauda pesada. O tamanho dos arquivos transferidos nas requisições HTTP pode ser modelado por distribuições de cauda pesada [13, 18, 23, 31]. Em cargas de serviço que seguem este tipo de distribuição, como é o caso da Web, uma pequena fração dos arquivos (os maiores) corresponde à maior parte da carga imposta ao servidor. Desta forma, requisições para arquivos grandes possuem à disposição a maior parte dos recursos do sistema, o que evita *starvation* destas requisições. Os resultados deste trabalho comprovam esta afirmação.

Considerando, portanto, a grande variabilidade dos tempos de processamento e a possibilidade de conhecer com antecedência o tamanho dos processos, a política mais efetiva para o tempo médio de resposta é a *SRPT (Shortest Remaining Processing Time)*. Comparações entre SRPT e PS no contexto específico dos servidores Web, feitas através de análise, simulação e experimentação, são apresentadas nos trabalhos [16, 18, 19]. Em todos os casos os resultados são bastante favoráveis à disciplina SRPT. Contudo, deve-se ressaltar que todas as avaliações foram realizadas em rede local. Avaliar um servidor Web em uma rede local pode ser muito diferente de avaliá-lo nas condições reais de conectividade da Internet. Numa rede local a latência é muito menor do que na Internet. Além disso, a banda disponível em uma rede local é normalmente muito maior e mais homogênea do que a banda observada na Internet. Para responder a perguntas do tipo "uma requisição de um arquivo pequeno feita por um usuário com conexão lenta deve ter maior ou menor prioridade do que a mesma requisição feita através de uma conexão rápida?" deve-se considerar as variações de conectividade na Internet.

O objetivo deste trabalho é propor uma política de escalonamento que reconheça e explore os benefícios das características apontadas: diversidade das condições de conectividade dos usuários na Internet; variabilidade dos tamanhos dos arquivos transferidos e possibilidade de conhecer com antecedência o tamanho dos processos. A política pode ser aplicada tanto em servidores Web como em Web *caches*, uma vez que ambos trabalham de maneira semelhante quanto ao tratamento de requisições estáticas. A idéia é explorar a variabilidade nas condições de conectividade dos usuários da Web. A proposta é dar prioridade para a requisição cuja conexão será encerrada primeiro, isto é, terá o menor tempo de duração. A conexão mais rápida é a que tem a menor requisição feita através da rede que tem as melhores condições de transmissão (maior

banda, menor latência, menor congestionamento). A política proposta foi denominada *Fastest Connection First* (FCF). Intuitivamente, uma política de escalonamento que priorize requisições originárias de conexões mais rápidas pode reduzir o tempo de serviço destas requisições, com conseqüente diminuição do tempo de duração das respectivas conexões e redução do número de requisições pendentes. Requisições pendentes são aquelas cujas conexões já poderiam ter sido encerradas pelo servidor, mas que ainda não foram concluídas devido à demora na transferência pela Internet. A finalização de uma conexão indica que os recursos do servidor estarão disponíveis para o atendimento de novas requisições.

Além disso, garantir maior prioridade para usuários com melhor conectividade auxilia o servidor a manter um tempo médio de resposta condizente com as expectativas dos usuários. Aqueles que possuem conexões mais velozes esperam respostas rápidas. Se a conexão é rápida, a requisição também será tratada rapidamente no servidor. O ponto fundamental desta política é garantir que os *buffers* das conexões mais rápidas recebam mais rapidamente os dados do sistema de arquivos. Quando as requisições são atendidas por servidores sobrecarregados, o tempo de serviço pode ser perceptível por um usuário que possui este tipo de conexão. Para usuários com conexões mais lentas, o tempo devido à espera no servidor é muito menos perceptível.

A política proposta foi comparada com outras duas políticas de escalonamento. A política padrão de servidores Web e a política SRPT (*Shortest Remaining Processing Time*). A avaliação foi feita através de simulação na qual foram modelados três componentes do ambiente Web, a saber: os usuários, o servidor Web e a Internet. Os usuários, que representam a carga de serviço imposta ao servidor, foram modelados a partir dos conceitos e fórmulas do gerador de cargas denominado SURGE [03, 23] (*Scalable URL Reference Generator*). Do servidor Web foram simulados os três principais recursos, CPU, disco e interface de rede. Para calcular o tempo de transmissão na Internet foi modelado o funcionamento dos protocolos TCP e HTTP, considerando as variações do RTT (*round trip time*), das condições de conectividade, incluindo congestionamentos e perdas de pacotes. Os resultados demonstraram que é possível reduzir o tempo médio de resposta do usuário melhorando-se a interação entre o servidor Web e a Internet.

Esta dissertação está organizada como descrito a seguir. O capítulo 2 apresenta um estudo detalhado dos protocolos da Web dando ênfase aos protocolos TCP e HTTP. Este estudo foi necessário para embasar a simulação do tempo de transmissão das

requisições HTTP na Internet. Este capítulo também apresenta o tratamento das requisições HTTP pelo servidor Web, desde sua admissão até o encerramento da conexão TCP. Esta descrição é a base para a construção do simulador do servidor e também para identificar flas onde a política de escalonamento pode ser aplicada quando implementada em um sistema real. No capítulo 3 são apresentados conceitos sobre diversas políticas de escalonamento uma vez que o objetivo deste trabalho é propor e avaliar políticas de escalonamento para servidores Web. O capítulo 4 apresenta a nova política de escalonamento. O capítulo 5 descreve a metodologia utilizada para avaliação de desempenho das políticas. O capítulo 6 apresenta os resultados e no último capítulo conclusões e trabalhos futuros são apresentados.

2. WEB E SEUS PROTOCOLOS

Neste capítulo é realizada uma descrição do funcionamento da Web, dos seus principais protocolos e de como ocorre a interação entre o servidor Web e a Internet. É descrita a forma como o servidor Web trata as requisições HTTP estáticas e, por fim, são apresentadas algumas métricas para avaliação de desempenho de servidores Web.

2.1 A Web

A *World Wide Web* ou simplesmente Web é um sistema de informação em hipertexto, hipermídia, distribuído, independente de plataforma, dinâmico, interativo e global. Sua arquitetura segue o modelo cliente/servidor e a rede de comunicação é a Internet. Este modelo permite a descentralização da Web e contribui para sua escalabilidade. Não há hierarquia para clientes e servidores, e nem mesmo um controle central.

O conceito de hipertexto foi proposto em 1945 por Vannevar Bush. Hipertexto é um modelo que organiza o conhecimento na forma de grafo onde os nós são unidades de informação, geralmente denominadas documentos, e as arestas são as ligações reais (*links*) entre nós semanticamente relacionados. Cada documento pode permitir o acesso a outros documentos implementando um *link* para os documentos referenciados. Assim, o conhecimento é organizado de forma não linear e o grafo pode ser percorrido de acordo com o interesse do leitor, que é incentivado a fazer uma exploração ativa do conteúdo. O processo de percorrer o hipertexto de acordo com interesses individuais recebe o nome de navegação. Documentos podem ser compostos por arquivos contendo texto, imagens, gráficos, áudio e vídeo, entre outras mídias. As conexões entre documentos de formatos diversos compõem o grande sistema hipermídia global que é a Web. Os termos hipertexto e hipermídia foram propostos por Ted Nelson em 1965.

Até o aparecimento da Web, as funcionalidades do modelo hipertexto tinham de ser incorporadas às aplicações. Um problema de pesquisa reconhecido era associar, na forma de hipertexto, informações e objetos armazenados em um mesmo computador porém gerenciados por aplicações diferentes. A inovação da Web com relação ao modelo hipertexto é dada pela ampliação da capacidade de endereçamento dos nós. A Web permite que os *links* sejam feitos não apenas entre documentos armazenados em

uma máquina, mas entre documentos armazenados em quaisquer máquinas conectadas à Internet, independente do tipo do documento, do sistema operacional ou do hardware. Os autores não precisam escrever sistemas de hipertextos. A facilidade de adicionar informação, ligando-a ao hipertexto global da mesma forma que se escreve uma nota marginal num texto em papel, é um atrativo essencial.

A Web foi idealizada por Tim Berners-Lee em março de 1989, como um meio de divulgação de pesquisas e novas idéias pelo CERN, Centro Europeu de Pesquisas em Física de Partículas, localizados em Genebra, na Suíça [33]. Foi desenvolvida para facilitar o gerenciamento dos vários projetos do CERN e permitir a interação e troca de idéias entre vários pesquisadores em lugares distintos. O primeiro programa para Web foi desenvolvido por Berners-Lee e colegas do CERN em novembro de 1990. Posteriormente diversos fabricantes desenvolveram programas que permitem a navegação na Web. Estes programas são chamados de navegadores ou *browsers*. Os *browsers* mais conhecidos são o Netscape e o Explorer, mas existem vários outros como o Mosaic, o Lynx, que é um *browser* somente para requisição de textos, e o HotJava, escrito em Java. O código de vários *browsers*, como o Netscape e o Lynx, está disponível na Web.

Quando o usuário seleciona um documento (normalmente clicando com o *mouse* sobre um *hyperlink*), o *browser* cria uma requisição e a envia para o servidor Web correspondente. A requisição inclui o nome do documento pedido, expresso como um URL (*Universal Resource Locators*), um conjunto de linhas de cabeçalho padrão HTTP, indicando que tipo de formato de dados o usuário aceitará, e informações de autenticação do usuário, que informa ao servidor se o usuário tem permissão para acessar o documento [31]. Uma vez que a requisição foi enviada, a máquina do usuário espera pela resposta. Quando a resposta chega, o *browser* analisa gramaticalmente a resposta. Dependendo da resposta, a máquina do usuário pode fazer novas requisições ao servidor, ou exibir o documento para visualização do usuário.

Para que o *browser* possa se conectar a um servidor Web ele precisa do endereço IP do servidor, que é um número de quatro bytes. Normalmente o usuário humano informa ou seleciona um URL contendo apenas o nome do servidor e não o endereço. Para descobrir o endereço IP do servidor o *browser* faz uma solicitação de tradução do nome a um servidor DNS (*Domain Name System*) local, que informa o endereço IP do servidor solicitado ou repassa a consulta para um nível mais alto da hierarquia de resolução de nomes [30]. Caso o sistema de resolução de nomes não consiga encontrar

um endereço associado ao nome do servidor então o *browser* exibe uma mensagem de erro ao usuário.

Na ARPANET (uma das redes que originou a Internet) não havia necessidade de servidores de DNS. Havia apenas um arquivo de *hosts* que era acessado em um *host* específico. Sempre que um novo *host* entrava na rede este arquivo precisava ser atualizado manualmente. Com o aumento do tamanho da rede esta prática tornou-se inviável. Para resolver este problema criou-se um sistema hierárquico de nomes de *hosts* baseado em domínios. Atualmente existem dois domínios de primeiro nível, a saber: genéricos e de países [33]. Os genéricos são os COM, EDU, MIL, GOV, NET, INT, entre outros. Os de países seguem a norma ISO 3166 na qual cada país deve ter um domínio, como por exemplo, BR, PT, IT, US, JP e NL. Dentro de cada domínio de primeiro nível podemos ter vários subdomínios, e assim sucessivamente. Os nomes de domínios são organizados na forma de uma árvore. Percorrendo-se esta árvore cada *host* tem seu nome definido pelo caminho até seu domínio de primeiro nível.

Quando um *browser* deseja obter o endereço IP de um *host* ele ativa um procedimento de biblioteca chamado *resolver*. Este procedimento envia um pacote UDP ao servidor de DNS local que é o mais próximo na árvore de domínios. Se este servidor não puder resolver o endereço IP do *host* solicitado, a consulta é repassada para o nível superior da árvore de domínios. Se a solicitação alcançar o primeiro nível da árvore sem ser atendida, ela pode ser direcionada para um subdomínio ao qual o *host* pesquisado pertença. Por exemplo, o servidor do primeiro nível sabe que qualquer *host* que termine com ".br" está no subdomínio BR. Desta forma, a solicitação desce a árvore de domínios até encontrar resposta. Caso ela não possa ser resolvida, uma mensagem de erro é retornada. Este tipo de pesquisa ao DNS é chamada de recursiva. Nem todos os servidores DNS utilizam pesquisa recursiva. Alguns, quando não conseguem resolver o IP do *host* pesquisado, apenas passam o endereço do próximo servidor de DNS a ser pesquisado.

2.2 Protocolos da Web

A comunicação entre usuário e servidor na Web envolve a operação conjunta de diversos protocolos que atuam em camadas e formam o que é conhecido como pilha de protocolos. Por exemplo, o HTTP executa sobre o TCP que, por sua vez, executa sobre o IP, que geralmente executa sobre *Ethernet* [27]. São dois os principais modelos de divisão de protocolos em camadas. O primeiro, tendo como base o trabalho da

International Organization for Standardization (ISO), é conhecido como *Reference Model of Open System Interconnection* da ISO, denominado modelo OSI da ISO. Este modelo contém sete camadas conceituais e foi criado para descrever protocolos para uma única rede [30]. O segundo modelo mais importante não surgiu através de uma comissão normatizadora, e sim de pesquisas que levaram à pilha de protocolos TCP/IP [30]. Os protocolos no modelo TCP/IP são organizados em quatro camadas conceituais construídas sobre uma quinta camada física de hardware. A Tabela 1 mostra as camadas, assim como alguns nomes de protocolos que atuam em cada camada.

Camada de Aplicação	HTTP, FTP, Telnet, SMTP, DNS
Camada de Transporte	TCP, UDP
Camada Inter-Rede	IP, ARP, RARP, ICMP, DHCP
Camada de Interface de Rede	*Ethernet*, *Token Ring*, FDDI, etc
HARDWARE	

Tabela 1 : Modelo conceitual TCP/IP.

Os protocolos mais diretamente ligados ao servidor Web são o IP, o TCP e o HTTP. Cada um deles é descrito nas subseções a seguir.

2.2.1 Camada Inter-Rede - IP (Internet Protocol)

A camada inter-rede, cujo protocolo utilizado na Internet é o IP, permite a comunicação entre máquinas de redes diferentes. Neste contexto, uma rede consiste de um segmento de cabo que interliga algumas máquinas. A função desta camada é criar a abstração de uma rede de redes de forma a permitir a comunicação através de várias redes e esconder a diversidade de tecnologias empregadas em cada rede componente. A camada inter-rede aceita um pedido para enviar um pacote originário da camada de transporte, juntamente com uma identificação da máquina para a qual o pacote deve ser enviado. A camada encapsula o pacote em um datagrama IP, preenche o cabeçalho do datagrama, utiliza o algoritmo de roteamento para decidir se entrega o datagrama diretamente ou o envia para um roteador e passa o datagrama para a camada de interface de rede [30].

As funções da camada inter-rede na Internet são realizadas pelo protocolo IP. Ele define um mecanismo de transmissão sem conexão, não-confiável e oferece três definições importantes. A primeira define a unidade básica de transferência de dados através de uma interligação em redes TCP/IP. A segunda, o software IP desempenha a

função de roteamento, escolhendo um caminho por onde os dados serão enviados. A terceira, além da especificação formal e precisa de formatos de dados e de roteamento, o IP inclui um conjunto de regras que concentram a idéia da entrega não-confiável de pacotes. As regras definem como os *hosts* e os roteadores devem processar os pacotes, como e quando as mensagens de erro devem ser geradas e as condições segundo as quais os pacotes podem ser descartados. As máquinas que executam o protocolo IP cooperam na base do melhor esforço: se for possível entregar uma mensagem, uma quantidade de esforço razoável é despendida na tentativa. Se a entrega não for possível, a mensagem é descartada.

2.2.2 Camada de Transporte - TCP (Transmission Control Protocol)

A primeira função da camada de transporte é prover a comunicação de um programa aplicativo para outro. Tal comunicação é chamada fim-a-fim. A camada de transporte pode regular o fluxo de informações. Ela pode fornecer transporte confiável, assegurando que os dados cheguem sem erros e em seqüência. Para isso, o protocolo de transporte faz com que o lado receptor envie confirmações e o lado transmissor retransmita pacotes perdidos. O software da camada de transporte divide o fluxo de dados transmitidos em pequenas partes e passa cada pacote, juntamente com o endereço de destino, à camada seguinte para ser transmitido [30].

As funções da camada de transporte na Internet são implementadas por dois protocolos, *o User Datagram Protocol* (UDP) e o *Transmission Control Protocol* (TCP). O protocolo UDP oferece transporte não-confiável entre duas máquinas na rede e opera na base da boa vontade. A funcionalidade que o UDP acrescenta à camada inter-rede é a capacidade de transportar mensagens oriundas de vários aplicativos diferentes através de um único canal de comunicação utilizando o conceito de portas. Se, devido a problemas de transmissão, mensagens forem perdidas, fica a cargo dos aplicativos a tarefa de recuperar a informação perdida. Para aplicativos que suportem a perda eventual de mensagens, o UDP propicia um meio de transporte eficiente e simples.

Se o aplicativo não tolera perda de mensagens, o protocolo TCP deve ser usado na comunicação. Na Internet, a transmissão de requisições HTTP geralmente ocorre sobre uma conexão TCP. O protocolo TCP garante a entrega ao destinatário, na ordem correta, de todas as mensagens enviadas pelo remetente. Além de transporte confiável, o TCP contém mecanismos que controlam fluxo de mensagens entre pares de máquinas, evitando assim que um produtor rápido inunde um consumidor lento com dados. O TCP

também contém mecanismos que controlam o tráfego de mensagens através da rede como um todo, evitando que a capacidade agregada da rede seja excedida. O protocolo especifica o formato dos dados e das confirmações que dois computadores trocam para oferecer uma transferência confiável, e também os procedimentos que são utilizados para assegurar que os dados cheguem corretamente. Ele especifica como o software TCP confirma múltiplos destinos em determinada máquina e como as máquinas recuperam-se de erros tais como pacotes duplicados ou perdidos. O protocolo também especifica como dois computadores iniciam uma transferência de *stream* TCP e como concordam sobre o término da transferência [30].

Embora a especificação TCP descreva como os programas aplicativos utilizam o TCP em termos gerais, ela não impõe os detalhes da interface entre o programa aplicativo e o TCP. Isso significa que a documentação do protocolo trata apenas das operações que o TCP oferece. Já que o TCP independe do sistema de comunicação físico, ele pode ser utilizado com uma variedade de sistemas de transmissão de pacotes. O TCP pode, por exemplo, ser implementado para utilizar linhas telefônicas por discagem, rede local, rede de fibra ótica de alta velocidade, ou uma rede de longa distância de velocidade mínima. Na realidade, um dos pontos fortes do TCP é a grande variedade de sistemas de transmissão que ele pode usar.

O serviço TCP é obtido através de pontos de acesso a serviços criados na origem e no destino, chamados *sockets*. Cada *socket* tem um número (endereço) de *socket*, que consiste do endereço IP do *host* e um número de 16 bits local ao *host* chamado porta. Uma porta pode ser visualizada como um ramal dentro de um endereço IP. Na arquitetura TCP/IP as portas são identificadas por números inteiros que variam entre 0 e 65535 (16 bits).

Para obter o serviço TCP, deve ser estabelecida uma conexão entre o *socket* da máquina que deseja enviar e da máquina que deve receber dados. Um *socket* pode ser utilizado por mais de uma conexão ao mesmo tempo. As conexões TCP são identificadas pelos *sockets* dos dois lados. Números de portas abaixo de 1024 são chamados *well-known ports* sendo reservados para serviços padronizados. Alguns exemplos são apresentados na Tabela 2.

Porta	Serviço
21	FTP
23	TELNET
25	SMTP
79	FINGER
70	GOPHER
80	HTTP
110	POP3
119	NNTP
194	IRC

Tabela 2 : Portas reservadas para serviços padronizados.

Todas as conexões TCP são *full-duplex*, ponto-a-ponto, não suportando *multicast* e *broadcast*. O cabeçalho tem um formato fixo de 20 bytes. O cabeçalho fixo pode ser seguido por opções de cabeçalho. Depois das opções, pode haver até 65515 bytes de dados. O TCP divide a seqüência de bytes a transmitir em segmentos. Os segmentos são a unidade básica de transferência na camada de transporte.

O TCP utiliza um mecanismo especializado de janela deslizante para resolver dois problemas importantes: transmissão eficaz e controle de fluxo. Com este mecanismo é possível enviar segmentos múltiplos antes que uma confirmação chegue. Isso aumenta o *throughtput* total, já que mantém a rede ocupada. A implementação de janela deslizante do TCP também resolve o problema de controle de fluxo fim-a-fim, ao permitir que o receptor restrinja a transmissão até que haja espaço suficiente no *buffer* para aceitar mais dados.

O mecanismo de janela deslizante TCP opera em nível de octetos (bytes), e não de pacotes ou segmentos. Os octetos do *stream* de dados são numerados seqüencialmente, e um transmissor mantém três ponteiros associados a cada conexão. O primeiro ponteiro marca a esquerda da janela deslizante, separando os octetos que tenham sido enviados e confirmados dos octetos ainda por enviar. Um segundo ponteiro marca a direita da janela deslizante e define o maior octeto da seqüência que pode ser enviado antes que mais confirmações sejam recebidas. O terceiro ponteiro marca o limite interno da janela que separa os octetos já enviados daqueles que ainda não o foram. O protocolo envia todos os octetos na janela sem intervalos de modo que o limite dentro da janela quase sempre segue da esquerda para a direita. Considerando que as conexões TCP são *full duplex*, duas transferências ocorrem simultaneamente em cada conexão, uma em cada direção. As transferências são completamente independentes pois em qualquer ocasião os dados podem fluir pela conexão em uma direção, ou em

ambas as direções. Assim, o software TCP de cada extremidade mantém duas janelas por conexão (para um total de quatro): uma desliza no *stream* de dados que está sendo enviado e a outra desliza à medida que os dados são recebidos.

O tamanho da janela deslizante TCP varia com o passar do tempo. Cada confirmação que especifica quantos octetos foram recebidos contém um notificador de janela que define quantos octetos de dados adicionais o receptor está preparado para aceitar. Desta forma, considera-se que o notificador da janela é o especificador do tamanho do *buffer* atual do receptor. Quando uma notificação de que a janela aumentou é recebida, o transmissor aumenta o tamanho de sua janela deslizante e continua a enviar octetos que não tenham sido confirmados. Quando a notificação é de que a janela foi diminuída, o transmissor reduz o tamanho de sua janela e interrompe o envio de octetos além do limite. O software TCP não deve contradizer notificações anteriores, encurtando a janela até posições anteriormente aceitáveis do *stream* de octetos. Em vez disso, notificadores menores acompanham confirmações, de modo que o tamanho da janela muda quando ela desliza.

A vantagem de utilizar uma janela de tamanho variável é que ela fornece o controle de fluxo e uma transferência confiável. Se os *buffers* do receptor começam a ficar cheios, a janela não poderá suportar mais pacotes, e então enviará um notificador de janela menor. Em casos extremos, o receptor indica um tamanho zero de janela para interromper todas as transmissões. Mais tarde, quando o espaço do *buffer* torna-se disponível, o receptor indicará um tamanho de janela diferente de zero para iniciar o fluxo de dados novamente.

Controlar o fluxo de dados na comunicação entre redes envolve dois problemas isolados. Primeiro, os protocolos de interconexão de dados precisam de um controle de fluxo fim-a-fim, entre a origem e o destino final. Segundo, protocolos de interconexão de dados precisam de um mecanismo de controle de fluxo que permita que sistemas intermediários (roteadores) controlem uma origem que envie mais dados do que a máquina possa tolerar. Quando máquinas intermediárias tornam-se sobrecarregadas a ponto de descartar pacotes, a condição é denominada congestionamento. O TCP utiliza o esquema de janela deslizante para resolver o problema de controle de fluxo fim-a-fim.

O TCP considera que congestionamento é uma condição de retardo longo causado por uma sobrecarga de datagramas em um ou mais pontos de comutação (p. ex. roteadores). Quando ocorre um congestionamento, os intervalos aumentam e o roteador começa enfileirar datagramas para distribuí-los mais tarde. Dependendo da taxa de

chegada, o roteador pode ter sua capacidade esgotada e datagramas começam a ser perdidos. A maioria dos protocolos de transporte utiliza os mecanismos de *timeout* e retransmissão para reagir a congestionamentos. As retransmissões somente agravam o congestionamento. O aumento de tráfego provoca o aumento do retardo, que provoca a elevação do tráfego, e assim por diante. Esta situação é conhecida como colapso de congestionamento.

Para evitar congestionamento, o TCP utiliza duas técnicas, a saber: início lento (*slow start*) e redução multiplicativa (*multiplicative decrease*) [30]. É utilizada uma nova janela denominada janela de congestionamento, semelhante à janela do receptor utilizada para controle de fluxo fim-a-fim. Desta forma a quantidade de dados que pode ser enviada fica limitada à menor entre as janelas de congestionamento e do receptor.

No início de uma conexão ou para recuperar-se de um congestionamento, o TCP utiliza a técnica de início lento (*slow start*). Quando a conexão está nesta fase, para cada segmento confirmado, dois novos segmentos podem ser enviados. Isto faz com que a janela de congestionamento aumente exponencialmente. Para evitar que este aumento gere novos congestionamentos o TCP impõe outra restrição. Quando a janela alcança a metade da janela original antes do congestionamento, o TCP entra numa fase de prevenção de congestionamento e diminui a taxa de incremento. Durante esta fase de prevenção, a janela é aumentada em apenas um segmento quando todos os segmentos da janela tiverem sido confirmados. Por outro lado, o mecanismo de redução multiplicativa faz com que o TCP, na ocorrência da perda de um segmento, reduza a janela de congestionamento pela metade até um mínimo de um MSS (*Maximum segment size*). Desta forma, para toda perda, o TCP reduz a janela de congestionamento exponencialmente. A modelagem do funcionamento do protocolo TCP é utilizada na seção 5.4 para calcular o tempo de transmissão das requisições HTTP na Internet.

Uma conexão TCP é estabelecida através de um *handshake* de três vias. O primeiro segmento é enviado por quem deseja abrir a conexão. Este segmento possui ligado o bit SYN (*synchronize sequence numbers* – sincronização de números de seqüência) do cabeçalho TCP. Além disso, este primeiro segmento possui um número de seqüência inicial ISN (*Initial Sequence Number*). Quando a estação remota recebe um segmento deste tipo, ela sabe que se trata da solicitação de abertura de conexão e devolve um segmento com o código SYN e um número seqüencial inicial de reconhecimento de segmento. A mensagem final de *handshake* (enviada por quem iniciou a conexão) é apenas uma confirmação. Ela simplesmente informa a estação

remota que ambos os lados concordam em que uma conexão foi estabelecida. Depois da conexão ter sido estabelecida, os dados podem fluir igualmente nas duas direções. Não há mestre ou escravo.

O *handshake* de três vias é necessário e suficiente para a sincronização correta entre as duas extremidades da conexão. Este protocolo de três vias cumpre duas funções importantes. Garante que ambas as extremidades estejam prontas para transferir dados, e permite que ambas concordem quanto aos números de seqüências iniciais. Os números de seqüência são enviados e confirmados durante o *handshake*. Cada estação precisa escolher aleatoriamente o número inicial que será usado para identificar bytes em um *stream* que estiver enviando. Os números de seqüência nem sempre começam com o mesmo valor.

Para verificar como os processos podem concordar em números de seqüência para dois *streams* após apenas três mensagens, o processo que inicia o *handshake*, chamado de *A*, passa seu número de seqüência inicial X no campo de seqüência do primeiro segmento SYN do *handshake* de três vias. O segundo processo, *B*, recebe o SYN, grava o número de seqüência inicial X e responde enviando seu número de seqüência inicial Y no campo de seqüência, bem como uma confirmação que especifica que *B* aguarda o próximo byte X+1. Na mensagem final do *handshake*, o processo *A* confirma ter recebido do processo *B* todos os bytes até Y. A latência decorrente do estabelecimento de conexão será modelada na seção 5.4.

Dois programas que utilizam o TCP para se comunicar podem também encerrar a conexão. Internamente o TCP utiliza o *handshake* de 3 vias modificado para encerrar conexões. Quando um programa aplicativo diz ao TCP que não há mais dados a serem enviados, o TCP fechará a conexão em uma direção. Para encerrar metade de uma conexão, o TCP transmissor completa a transmissão dos dados restantes, espera que o receptor os confirme e, a seguir, envia um segmento com o conjunto de bits FIN. O TCP receptor confirma com o segmento FIN e informa ao programa aplicativo de sua extremidade que não há dados disponíveis.

Quando uma conexão tiver sido concluída em determinada direção, o TCP se recusará a aceitar mais dados para essa direção. Enquanto isso, os dados podem continuar a fluir na direção oposta até que o transmissor a encerre. Quando ambas as direções tiverem sido concluídas, o software TCP utilizará um *handshake* de três vias modificado para encerrar a conexão.

A diferença entre o *handshake* de três vias usado para estabelecer e interromper conexões ocorre depois que uma estação recebe um segmento FIN inicial. Em vez de gerar um segundo segmento FIN imediatamente, o TCP envia uma confirmação e, a seguir, informa ao aplicativo sobre a solicitação de encerrar a conexão. Desde informar ao programa aplicativo sobre a solicitação até obter uma resposta pode levar um tempo considerável. A confirmação evita a retransmissão do segmento FIN inicial durante a espera. Finalmente, quando o programa aplicativo instrui o TCP para encerrar a conexão, o TCP envia o segundo segmento FIN e a estação destino responde com uma terceira mensagem.

O TCP foi projetado a partir dos pressupostos de que as conexões seriam infreqüentes e a quantidade de bytes transferida em cada conexão seria grande. A ênfase do projeto foi em correção e completeza da conexão e não em desempenho. Esses pressupostos não são válidos para o ambiente Web que requer, tipicamente, muitas conexões de curta duração em seqüência, cada uma transmitindo poucas centenas de bytes [22]. O tamanho médio das transmissões da Web gira em torno de 12 Kbytes com mediana em torno de 3 Kbytes [15]. O estabelecimento de uma conexão TCP inclui uma troca de segmentos entre usuário e servidor, procedimento que gera uma latência relativamente pequena em transmissões longas mas considerável em transmissões pequenas [50]. O mecanismo de conexões persistentes, disponível na versão 1.1 do protocolo HTTP, reduz a penalidade gerada pela latência inerente ao estabelecimento de cada conexão [07]. Contudo, trabalhos recentes [12, 34, 45] indicam que a maioria das requisições HTTP são realizadas através da versão 1.0. Além disso, a eficiência da versão 1.1 ainda é discutível como indicam os experimentos realizados em [20].

2.2.3 Camada de Aplicação - HTTP (HyperText Transfer Protocol)

No nível mais alto da pilha de protocolos são executados programas aplicativos que acessam serviços disponíveis através de uma interligação em redes TCP/IP. Um aplicativo interage com um dos protocolos da camada de transporte para enviar ou receber dados. Cada programa aplicativo escolhe o método de transporte necessário, que tanto pode ser uma seqüência de mensagens individuais ou um *stream* contínuo de bytes. O programa aplicativo passa para a camada de transporte os dados na forma adequada, para que possam, então, ser transmitidos. Servidores Web podem executar os protocolos HTTP, FTP, SMTP e outros. Porém, considerando que o objetivo desta

dissertação é avaliar o desempenho do servidor Web no atendimento de requisições HTTP, apenas este protocolo será descrito.

O HTTP é um protocolo da camada de aplicação, que reconhece e transfere dados em sistemas hipermídia distribuídos. Os dados transferidos podem ser de formatos diversos como texto, imagens, hipertexto, áudio ou qualquer outro formato [33].

Formatos proprietários podem ser definidos sem necessidade de padronização. No entanto, é necessária uma negociação entre usuário e servidor sobre o formato da resposta pois o usuário pode não estar apto a tratar todos os tipos de formato. Na Internet, a comunicação via HTTP geralmente ocorre sobre uma conexão TCP. O protocolo HTTP é independente de estado, isto é, os objetos são retornados com base no URL e de forma independente de qualquer operação prévia realizada pelo usuário. O protocolo não armazena informações sobre requisições anteriores, não definindo, portanto, o conceito de estado. O HTTP tem sido utilizado como o protocolo da Web desde 1990 e é responsável pela maior parte do tráfego nos *backbones* da Internet [22]. As versões do HTTP utilizadas atualmente são a 1.0 e a 1.1. As duas versões apresentam diferenças significativas que serão discutidas no final desta seção. O HTTP é um protocolo genérico utilizado também para comunicação entre usuários e servidores de outros protocolos tais como SMTP (*Simple Mail Transfer Protocol*), NNTP (*Network News Transfer Protocol*) e FTP (*File Transfer Protocol*). Isto significa que o HTTP provê acesso hipermídia aos recursos disponibilizados por diversas aplicações, permitindo assim que o usuário, utilizando apenas um *browser*, acesse os serviços providos pelos outros protocolos que o HTTP engloba. O protocolo HTTP opera na forma requisição-resposta e é utilizado para comunicação entre usuários e servidores. Um usuário estabelece uma conexão com um servidor. Após ser estabelecida a conexão, o usuário envia a requisição ao servidor. Uma requisição consiste de um método a ser aplicado a um recurso do servidor, identificado pelo URL, a versão do protocolo e outras informações. O servidor processa a requisição e envia a resposta que contém um código de erro ou sucesso, possivelmente o recurso pedido e outras informações. Dois tipos de mensagens são definidos no protocolo HTTP: requisições do usuário para o servidor e respostas do servidor para o usuário. Ambos os tipos de mensagem consistem de um conjunto de campos que compõem o cabeçalho da mensagem e, em alguns casos, do conteúdo da mensagem. Existem cabeçalhos específicos para mensagens de

requisição, para as mensagens de resposta e também cabeçalhos para a parte do conteúdo da mensagem.

Os campos do cabeçalho de uma mensagem de requisição permitem ao usuário passar para o servidor informações sobre a requisição e sobre o próprio usuário. Estes campos atuam como modificadores da requisição, com semântica equivalente aos parâmetros das funções em linguagens de programação. Os campos do cabeçalho de uma mensagem de resposta servem para o servidor passar para o usuário informação adicional sobre a resposta, sobre o servidor e sobre acessos adicionais ao recurso identificado pelo URL na requisição. Informação sobre o próprio recurso é transferida em campos do cabeçalho do conteúdo da mensagem. Uma mensagem de requisição inclui o método a ser aplicado ao recurso, o identificador do recurso e a versão do protocolo HTTP. O método é o código de uma operação a ser executada sobre o recurso identificado pelo URL. GET, HEAD e POST são os métodos mais comuns.

Após receber e interpretar a mensagem de requisição, o servidor responde com uma mensagem de resposta HTTP. A primeira linha da resposta indica a versão do protocolo e um código numérico de três dígitos acompanhado de uma frase descritiva do código. O código indica o resultado da tentativa de entender e satisfazer a requisição. A frase é uma descrição textual curta do código para o usuário humano. Exemplos de códigos e suas respectivas frases estão na Tabela 3.

Código	Descrição
200	OK
304	Not Modified
400	Bad Request
403	Forbidden
404	Not Found

Tabela 3 : Código de resposta de requisições HTTP.

Na versão 1.0 do HTTP, a conexão é encerrada pelo servidor logo após o envio da resposta. Nesta versão uma nova conexão é aberta para buscar cada URL requisitado. O uso de imagens e outros dados associados a um URL requerem, portanto, que o usuário estabeleça múltiplas conexões com o servidor em um curto espaço de tempo, o que aumenta a carga sobre os servidores e a Internet. A versão 1.1 implementa o conceito de conexões persistentes, mecanismo que permite a utilização de uma mesma conexão para o atendimento de mais de uma requisição. Esta estratégia resulta em melhor desempenho em vários dos sistemas envolvidos na conexão [50]. As conexões

podem ser encerradas por uma variedade de razões. Qualquer parte pode encerrar prematuramente a conexão devido à ação do usuário, expiração (*timeout*), falha no programa e outras razões. O fechamento da conexão por qualquer parte sempre encerra a requisição corrente independente do seu estado. A versão 1.1 do protocolo HTTP apresenta várias modificações e extensões em relação à versão 1.0. O objetivo principal das mudanças foi melhorar o desempenho, reduzindo a latência de transferência dos documentos e o tráfego na rede. A mudança mais importante foi a definição do mecanismo de conexões persistentes. Outras mudanças estão relacionadas a mecanismos de *cache*, gerência de conexão, transmissão de mensagem, negociação do conteúdo e procedimentos para redução na utilização da rede.

2.3 Tratamento das Requisições HTTP em Servidores Web

Na sua forma mais simples, uma requisição Web envolve o estabelecimento de apenas uma conexão direta entre usuário e servidor. Seguindo o modelo cliente/servidor, o usuário sempre inicia uma interação pedindo ao servidor uma página. O software servidor executa continuamente, esperando as requisições dos usuários. Ao receber uma requisição, o servidor identifica o arquivo requisitado, procura o arquivo no seu espaço de armazenamento e o envia para o *browser* pela mesma conexão. O conceito de servidor Web pode ser ampliado para englobar o ambiente composto pelo computador, sistema operacional e o próprio programa servidor, mais o conjunto de documentos e aplicações disponíveis no lado do servidor.

Pelo menos dois tipos de intermediários são comuns no caminho entre o usuário e o servidor: *proxy* e *gateway*. Um *proxy* é um programa intermediário que atua como servidor e também como cliente com o objetivo de fazer requisições em nome dos usuários. As requisições feitas pelos usuários ao *proxy* são servidas por sistemas de *cache* associados ou são reenviadas com possível tradução para outros servidores. Um *proxy* deve interpretar e, se necessário, reescrever a mensagem de requisição antes de reenviá-la. *Proxies* são freqüentemente utilizados como portais do lado do usuário em sistemas de *firewalls* da rede na qual o usuário está conectado. Um *gateway* é um servidor que age como um intermediário de outro servidor. O *gateway* recebe a requisição como se fosse o servidor destino da requisição. O usuário pode não estar ciente de que a requisição está sendo tratada por um *gateway*. *Gateways* são freqüentemente utilizados como portais do lado do servidor.

Os servidores Web processam requisições em paralelo, ou seja, várias requisições são atendidas concorrentemente. Diversas estratégias são utilizadas para implementar o paralelismo: manter um conjunto (*pool*) de processos servidores, criar um novo processo (*fork*) ou gerar uma *thread* para cada requisição nova, manter um conjunto de *threads* ou utilizar servidor de entrada e saída assíncrono para tratar as conexões em paralelo com um único processo. Estas estratégias podem ser combinadas, gerando um modelo misto [01].

A estratégia de criação de um novo processo é a mais simples mas é computacionalmente cara para a carga Web, cuja taxa de chegada de requisições é alta e as requisições têm curta duração [19]. Muito tempo seria gasto para criar novos processos. As demais estratégias são utilizadas pelos diversos servidores Web disponíveis. Por exemplo, o servidor Apache mantém um conjunto de processos servidores (*preforking*) e o Netscape Enterprise Server é *multithreaded*. O hardware do servidor precisa ser mais potente do que o hardware do usuário pois o servidor deve ser capaz de atender vários usuários simultaneamente. O fator limitante do desempenho do servidor pode ser a interface de rede, o disco e/ou a memória. Por isso é recomendado que o servidor tenha um boa interface de rede, discos de acesso rápido e muita memória.

O sistema operacional também influencia o desempenho do servidor, especialmente quando este é submetido a cargas elevadas. A principal diferença entre os sistemas operacionais Unix e Windows NT, sob o aspecto dos servidores Web, é a escalabilidade. Estações de trabalho com Unix formam um ambiente muito mais escalável do que PCs com Windows NT. Existem dezenas de servidores Web disponíveis. O Apache é, com ampla vantagem, o servidor Web mais utilizado em todo o mundo e isso se deve, em grande parte, ao fato de ser gratuito. Atualmente o Apache é utilizado em mais da metade dos servidores existentes na Internet. Outros servidores populares são o IIS da Microsoft, o JavaWebServer da Sun, o JigSaw, o NCSA e o Enterprise Server da Netscape.

A latência da rede tem grande impacto sobre o tempo de resposta de um servidor Web, de acordo com [05]. Este artigo apresenta testes com cargas de trabalho em ambientes que simulam uma Intranet, praticamente sem latência de rede, e a Internet com suas variações de latência. O problema das grandes latências é que o servidor precisa manter um processo ocupado com a requisição até que o último pacote de dados seja enviado ao usuário. Os resultados apresentados no trabalho [05] reforçam a idéia da política proposta nesta dissertação no sentido de dar prioridade às requisições de

conexões que tenham menor latência na rede. Ao permitir que estas requisições sejam concluídas mais rapidamente, reduz-se o impacto que a latência da Internet provoca no desempenho do servidor.

O trabalho [42] apresenta uma avaliação da interação entre o servidor Web e a Internet. Foram avaliadas quatro combinações de carga de serviço variando-se a carga da rede e do servidor. Os resultados demonstram que quando o servidor está sobrecarregado e a rede subtilizada, o tempo de resposta duplica, podendo alcançar, para alguns casos, até seis vezes o tempo de resposta observado nas situações em que o servidor não está sobrecarregado. Observou-se também que a maior parte deste tempo adicional é gasto na fila de conexões já iniciadas que aguardam recursos para serem processadas. No servidor Web existem duas filas de conexões TCP [34]. Na primeira fila ficam as conexões que estão em fase de abertura trocando segmentos de sincronização. Na segunda fila ficam as conexões já iniciadas prontas para serem atendidas. Quando o servidor está sobrecarregado, as conexões terminam a fase de sincronização e são passadas para a segunda fila onde aguardam pela liberação de recursos do servidor. O tempo de espera nesta fila faz com que o tempo de resposta final seja o dobro do tempo observado quando servidor está com um carga de serviço menor. Embora os resultados apresentados no experimento [42] tenham permitido determinar a parcela de contribuição do servidor no tempo de resposta final, não foi possível, através deles, determinar qual a influência da rede sobre o desempenho do servidor. Outro ponto não avaliado no experimento foi a possibilidade de haver conexões com diferentes larguras de banda e não apenas diferentes RTTs. A existência de diferentes larguras de banda, RTTs, e a ocorrência de perdas de segmentos podem provocar um comportamento do servidor bastante diferente do observado, o que não foi avaliado em nenhum dos trabalhos encontrados.

O trabalho [34] apresenta uma avaliação comparativa do desempenho de um servidor Web em uma rede local e na Internet. Ao adicionar um RTT de 200 milissegundos para simular a Internet, o desempenho do servidor Web foi reduzido em 54%. A redução no desempenho, segundo os autores, foi devido principalmente à existência de um maior número de conexões abertas simultaneamente uma vez que a transmissão na Internet se prolonga por um tempo maior do que aquele observado em uma rede local. Este número mais elevado de conexões abertas reduz o desempenho do servidor porque exige maior quantidade de memória e mais trocas de contexto entre processos. No caso do servidor Web Apache (servidor avaliado), outro fator que

contribuiu para reduzir o desempenho foi o limite do número de processos ativos. No Apache, o limite normalmente utilizado é de 152 processos. A comprovação da existência deste tipo de problema em servidores Web ajuda a justificar a política proposta. Utilizando-se a política FCF a tendência, como será demonstrado, é de concluir mais rapidamente as requisições de conexões rápidas e, portanto, reduzir este problema de desempenho.

2.3.1 Respostas de Conteúdo Estático ou Dinâmico

A resposta que um servidor Web envia para um usuário pode corresponder a um conteúdo estático ou a um conteúdo gerado dinamicamente. Uma resposta de conteúdo estático corresponde a um arquivo lido a partir do sistema de arquivos do servidor. Este arquivo podem estar nos mais diversos formatos, entre os quais: arquivos HTML, imagens, áudio, vídeo, arquivos formatados (postscript, pdf, doc), arquivos binários (*applets*), entre outros. São arquivos disponíveis no diretório padrão, em subdiretórios ou ainda em diretórios de usuários. O custo de processamento de uma requisição estática corresponde à leitura do arquivo solicitado o envio pela conexão TCP aberta.

Respostas de conteúdo dinâmico são geradas em tempo real em função de parâmetros enviados na requisição. Estas respostas são compostas, em sua maioria, de resultados de consultas a banco de dados, de resultados de processamento, ou de atualizações feitas no servidor. Elas são geradas por aplicações ou código executável referenciado nas páginas Web. Este código é executado no servidor para gerar a resposta, que normalmente é um arquivo no formato HTML, que será enviado para o usuário. Isto permite aos projetistas mudar a apresentação do *site* sem precisar se preocupar com o conteúdo, já que este será gerado dinamicamente.

Existem diversas tecnologias que permitem criar conteúdo dinâmico na Web, entre elas estão programas CGI (*Common Gateway Interface*), aplicações ISAPI (*Internet Server* API), NSAPI (*Netscape Server* API) e servidores especializados que interpretam linguagens específicas como ASP (*Active Server Pages*), PHP (*Personal Home Page*) e JSP (*Java Server Pages*). As tecnologias como *JavaScript, Vbscript* e *Applet Java* permitem que parte do código seja executado na máquina do usuário, o que torna ainda mais dinâmico o conteúdo da página e ajuda a reduzir o tempo de resposta evitando a latência de rede e do servidor.

O padrão *Common Gateway Interface* foi o primeiro método utilizado para incorporar conteúdo dinâmico em páginas Web. Ele é o mecanismo de comunicação

entre o servidor Web e o programa ou aplicação que gera o conteúdo dinâmico para composição da página. Aplicações CGI são, por sua vez, programas ou *scripts* que recebem dados de entrada através da requisição HTTP do usuário e retornam resultados para o servidor que os transmite de volta para o usuário.

As aplicações CGI são sempre executadas na máquina servidora. Cada requisição a um serviço dinâmico acarreta a criação de um novo processo no servidor para gerar e tornar disponível a informação desejada. A necessidade de gerar um novo processo é um fator limitante do desempenho deste tipo de mecanismo. Para superar este problema surgiu uma nova interface chamada *FastCGI*. A diferença desta interface é que os processos são mantidos de forma permanente de modo a atender múltiplas requisições e com isso reduzir o *overhead* causado pela criação e destruição de processos. Além do padrão CGI, os demais mecanismos utilizados para gerar respostas de conteúdo dinâmico são considerados extensões e estão em constante evolução. Como este trabalho se destina a avaliar o desempenho de servidores Web atendendo requisições estáticas estas extensões não foram estudadas.

Seja uma resposta de conteúdo estático ou dinâmico, para que as requisições HTTP possam ser transmitidas através do TCP é necessária a abertura de uma conexão. Uma vez estabelecida a conexão, a estação cliente envia uma mensagem com a requisição feita pelo usuário, representada pelo seu URL. Antes de atingir o meio físico de transmissão, um ou mais segmentos TCP são montados e submetidos à camada de inter-rede. Neste ponto, um ou mais datagramas IP são montados e submetidos à interface de rede. Cada um destes datagramas chega à interface de rede da estação servidora causando uma interrupção. Esta interrupção é tratada pelo sistema operacional que extrai a mensagem HTTP e a envia para a porta dedicada ao servidor Web. Este realiza um processamento (*parsing*) sobre a mensagem recebida para saber qual é a informação desejada e verifica, a partir de informações no cabeçalho da mensagem, se ele pode atender a requisição feita, baseado-se em permissões de acesso, autorização e autenticação. A partir deste ponto começa o processamento da requisição. O servidor procura pelo arquivo solicitado no seu *cache* local, caso não o encontre, transferências de disco são realizadas. No caso de páginas dinâmicas, os processos que geram os dados requisitados são criados e executados. Uma mensagem HTTP de resposta é, então, construída e enviada ao usuário. Alguns servidores fazem, logo após o envio da resposta ao usuário, um registro (*logging*) para a requisição tratada. Este registro é gravado em arquivos de *log*.

2.3.2 Tratamento da Entrada de Pacotes de Requisições HTTP

Os pacotes contendo as solicitações ou respostas HTTP chegam aleatoriamente na estação destino (servidor ou usuário). Para tratar esta chegada aleatória de pacotes, alguns computadores utilizam o mecanismo de interrupção de software. Quando um pacote chega, uma interrupção de hardware ocorre e o *driver* de dispositivo efetua suas tarefas usuais de aceitar o pacote e reinicializar o dispositivo. Antes de retornar da interrupção, o *driver* de dispositivo solicita que o hardware programe a execução de uma segunda interrupção, a qual terá prioridade mais baixa. Assim que a interrupção de hardware é concluída, a interrupção de baixa prioridade ocorre exatamente como se um outro dispositivo de hardware tivesse efetuado uma interrupção. Essa segunda interrupção, conhecida como interrupção de software, suspende o processamento e faz a CPU saltar para o código que fará o tratamento da interrupção. Desse modo, em alguns sistemas, todo o processamento de entrada ocorre como uma série de interrupções. A idéia foi formalizada em um mecanismo do *System* V do UNIX, conhecido como *streams* [27].

Como a entrada ocorre em tempo de interrupção, o código do *driver* de dispositivo não pode chamar procedimentos arbitrários para processar cada pacote; o sistema precisa ser projetado para retornar rapidamente de uma interrupção. Portanto, o procedimento de interrupção não chama o software do IP diretamente. Além disso, como o sistema utiliza um processo isolado para implementar o IP, o *driver* de dispositivo não pode chamar o IP diretamente. Em vez disso, para sincronizar a comunicação, o sistema emprega uma fila em conjunto com primitivas de passagem de mensagens. Quando um pacote que transporta um datagrama IP chega, o software de interrupção precisa colocar o pacote em uma fila e passar uma mensagem para comunicar o processo IP de que um pacote chegou. Quando não tem pacotes para tratar, o processo IP fica aguardando a chegada de datagramas. Há uma fila de entrada associada a cada dispositivo de rede; um só processo IP extrai datagramas de todas as filas e os processa.

Ao aceitar um datagrama que chega, o processo IP precisa decidir para onde enviar tal pacote para processamento adicional. Se o datagrama transporta um segmento TCP, precisa ir para o módulo TCP; se transporta um datagrama UDP, precisa ir para o módulo UDP.

Em virtude da complexidade do TCP, a maioria dos sistemas utiliza um processo separado para gerenciar segmentos TCP que chegam. Uma conseqüência da existência

de processos IP e TCP separados é que eles precisam usar um mecanismo de comunicação entre processos quando interagem. O IP chama um procedimento para depositar segmentos em uma porta de comunicação e o TCP usa outro procedimento para recuperá-los.

Depois de receber um segmento, o TCP utiliza os números de porta para encontrar a conexão à qual o segmento pertence. Caso o segmento contenha dados, o TCP irá inserir os dados em um *buffer* associado à conexão e retornar uma mensagem de confirmação para o transmissor. Caso o pacote que chega transporte uma mensagem de confirmação, o processo de entrada do TCP precisará também se comunicar com o processo de temporização (*timer*) do TCP para cancelar a retransmissão pendente.

2.3.3 Tratamento da Saída de Pacotes de Requisições HTTP

Pacotes de saída podem surgir por duas razões. Ou um programa aplicativo passa dados para um dos protocolos de alto nível, o qual, por sua vez, envia uma mensagem para um protocolo de nível inferior e, por fim, gera transmissão em uma rede, ou o software de protocolo contido no sistema operacional transmite informações (p. ex., uma mensagem de confirmação ou uma resposta a uma solicitação de eco). Nos dois casos, um pacote precisa ser enviado por uma determinada interface de rede.

Para ajudar a isolar a transmissão de pacotes da execução de processos que implementem programas aplicativos e protocolos, o sistema possui uma fila de saída separada para cada interface de rede.

As filas associadas a dispositivos de saída representam uma parte importante do projeto do servidor. Elas permitem que processos gerem um pacote, coloquem-no em uma fila de saída e continuem a execução sem esperar que o pacote seja enviado. Enquanto isso, o hardware pode continuar a transmitir pacotes. Se o hardware estiver inativo quando um pacote chegar, o processo que estiver executando uma saída colocará seu pacote em uma fila e chamará uma rotina *driver* de dispositivo para inicializar o hardware. Quando a operação de saída é concluída, o hardware interrompe a CPU. A rotina de processamento de interrupções, que faz parte do *driver* de dispositivo, retira da fila o pacote que acaba de ser enviado. Caso haja mais pacotes na fila, a rotina de processamento de interrupções reinicia o hardware para enviar o próximo pacote. Então, a rotina de processamento de interrupções retorna da interrupção, permitindo que o processamento normal continue. Assim, do ponto de vista do processo IP, a transmissão de pacotes corre automaticamente em segundo plano. Enquanto houver pacotes em uma

fila, o hardware continuará a transmití-los. O hardware só precisa ser inicializado quando o IP deposita um pacote em uma fila vazia.

Evidentemente, cada fila de saída possui capacidade limitada, e pode ficar lotada se o sistema gerar pacotes mais rapidamente que o hardware da rede pode transmití-los. Presume-se que tais casos sejam raros, mas se efetivamente ocorrerem, processos que gerem pacotes precisarão fazer uma escolha: descartar o pacote ou efetuar um bloqueio até que o hardware conclua a transmissão de um pacote e libere mais espaço [27].

2.3.4 Tratamento de Pacotes Passando do TCP pelo IP até a Saída na Rede

Assim como a entrada, a saída do TCP é complexa. Conexões precisam ser estabelecidas, dados precisam ser dispostos em segmentos, e os segmentos precisam ser retransmitidos até que as mensagens de confirmação cheguem. Uma vez colocado em um datagrama, um segmento pode ser passado ao IP para roteamento e entrega. O software emprega dois processos do TCP para administrar a complexidade. O primeiro, denominado *tcpout*, gerencia a maior parte dos detalhes de segmentação e transmissão de dados. O segundo, denominado *tcptimer*, programa a execução de *timeouts* de retransmissão e avisa ao *tcpout* quando um segmento precisa ser retransmitido [27].

O processo *tcpout* usa uma porta para sincronizar entradas provenientes de vários processos. O TCP se baseia em *streams*, o que significa que ele permite a um programa aplicativo enviar alguns bytes de dados por vez. Consequentemente, os itens encontrados na porta não correspondem a pacotes ou segmentos individuais. Em vez disso, um processo que emite dados os insere em um *buffer* de saída e coloca uma só mensagem na porta, comunicando o TCP de que mais dados foram escritos. O processo *tcptimer* deposita uma mensagem na porta sempre que um *timer* expira e o TCP precisa retransmitir um segmento. Assim, podemos imaginar a porta como uma fila de eventos a serem processados pelo TCP - cada evento pode causar transmissão ou retransmissão de um segmento.

Depois de produzir um datagrama, o TCP passa-o ao IP para entrega. O IP escolhe uma interface de rede pela qual o datagrama precisar ser enviado e passa o datagrama ao processo de saída de rede correspondente. A figura 1 ilustra o caminho de dados que saem do TCP.

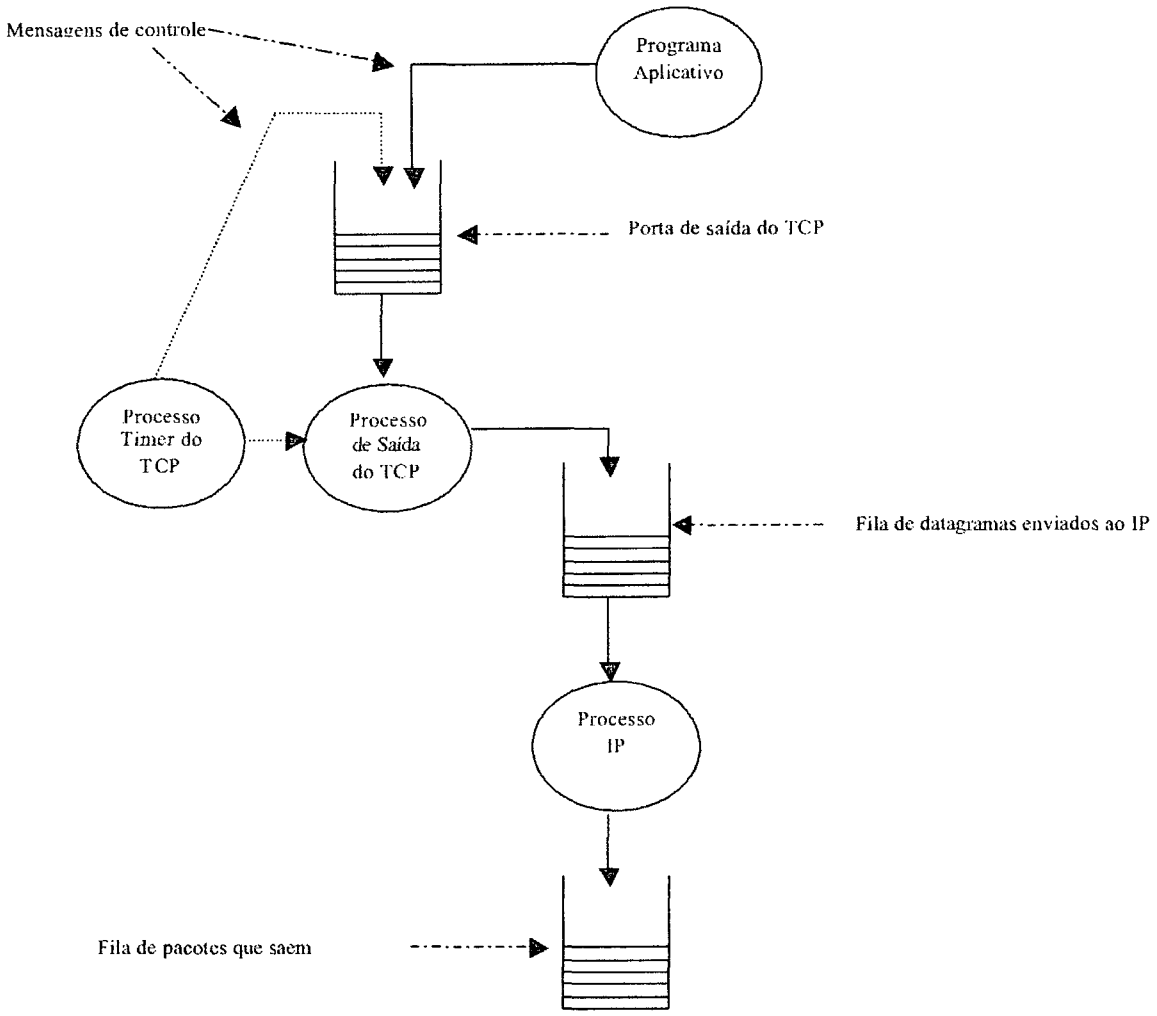

Figura 1 : Controle de fluxo na saída de dados no TCP.

2.3.5 Métricas de Desempenho em Servidores Web

As métricas mais utilizadas para quantificar o desempenho de servidores Web são a taxa de serviço e o tempo de resposta [50]. A taxa de serviço mede a freqüência de execução de operações, ou seja, o número de operações completadas por unidade de tempo. Pode ser medida em Megabits por segundo, número de conexões realizadas por segundo ou número de requisições HTTP por segundo.

O tempo de resposta do servidor para uma requisição corresponde ao intervalo de tempo desde o instante em que a requisição chega à interface de rede até o momento em que o servidor libera a informação para o meio físico. Contudo, no ambiente Web, pode haver diversos conceitos de tempo de resposta. O tempo de resposta observado pelo usuário tem início com a submissão do URL e termina com a apresentação dos dados no monitor de vídeo do usuário. Neste caso, diversos componentes do sistema contribuem para o tempo de resposta, dentre os quais, o servidor Web, a rede e a máquina do próprio usuário. A máquina do usuário impõe um atraso devido ao tempo gasto pelo *browser* para exibir o documento solicitado. A latência de rede representa o tempo gasto para transferir o pedido HTTP do usuário até o servidor, e posteriormente transferir a resposta HTTP do servidor para o usuário. Está incluído na latência de rede o tempo necessário para fazer o *handshake* de três vias para abrir a conexão TCP. Cuidado especial deve-se ter ao computar o tempo de abertura de conexão quando é utilizada a versão 1.1 do protocolo HTTP. Nesta versão diversas requisições podem ser transmitidas com o custo de abertura de apenas uma conexão. No servidor, o atraso é devido ao tempo de processamento da requisição. O tempo de processamento no servidor pode ser dividido entre os diversos componentes do sistema, tais como: processador, disco e interface de rede.

As métricas de desempenho computacional também podem ser gerais ou especializadas. As gerais quantificam o desempenho global do servidor. As especializadas quantificam o desempenho de componentes específicos, como o processador, disco, interface de rede, sendo normalmente utilizadas para verificar se a capacidade de um dado componente está adequada.

3. ESCALONAMENTO DE PROCESSOS

Tendo em vista que esta dissertação tem por objetivo propor uma nova política de escalonamento para servidores Web, este capítulo apresenta uma revisão dos conceitos acerca deste assunto. As principais políticas de escalonamento são apresentadas e discutidas tendo por objetivo definir sua aplicabilidade, seus pontos fortes e suas deficiências.

3.1 Conceitos e Critérios

Escalonar consiste em atribuir processos a processadores e determinar em que ordem estes processos serão executados. O escalonamento é de importância vital em sistemas paralelos e distribuídos, podendo ser considerado um dos problemas mais desafiantes nesta área. O problema básico de escalonamento é satisfazer simultaneamente objetivos conflitantes como obter tempo de resposta rápido, aumentar o *throughput*, tratar os processos com justiça, evitar *starvation* e conciliar processos de alta prioridade com baixa prioridade. O conjunto de regras utilizado para determinar como, quando e qual processo deverá ser executado é conhecido como política de escalonamento.

Todo software executado em um computador, incluindo o próprio sistema operacional, é organizado como um conjunto de processos seqüenciais, ou simplesmente processos [36]. Um processo é um programa em execução, incluindo os valores correntes de todos os registradores do hardware e das variáveis alteradas por ele no curso de sua execução. Principalmente em sistemas multiprogramados e/ou multiprocessados, o conceito de processo é utilizado por conveniência na execução concorrente de tarefas. Conceitualmente, cada processo possui o seu próprio processador virtual. Na realidade, o que ocorre é o chaveamento do processador entre cada um dos processos dando a impressão que eles estão executando em paralelo (pseudoparalelismo). O chaveamento rápido do processador entre diversos programas em execução (processos) é denominado multiprogramação.

Em um sistema onde vários processos são atendidos paralelamente, há situações nas quais dois ou mais processos estarão prontos para executar, ou seja, não estarão esperando por entrada ou saída ou por outra tarefa. Quando há mais de um processo nesta situação, o sistema operacional deve decidir qual deles vai executar primeiro. A

parte do sistema operacional a quem cabe tomar esta decisão é denominada escalonador e o algoritmo utilizado em sua programação é chamado de algoritmo de escalonamento.

Um servidor Web é um sistema multiprogramado com capacidade para atender concorrentemente várias requisições. O atendimento concorrente de requisições pode ser implementado de várias formas: mantendo um conjunto (*pool*) de processos servidores, criando um novo processo (*fork*) ou gerando uma *thread* para cada requisição nova, mantendo um conjunto de *threads* ou utilizando servidor de entrada e saída assíncrono para tratar as conexões em paralelo com um único processo. Seja qual for o mecanismo utilizado, o escalonador do sistema operacional precisa decidir qual requisição ou processo receberá a atenção do processador em determinado momento. A política de escalonamento, define quando e qual requisição será atendida. O escalonador do sistema deve decidir, baseado em um conjunto de regras, qual processo deve ter a atenção do processador. Vários critérios de decisão são conhecidos para algoritmos de escalonamento [37]:

- **Taxa de serviço** - É o número de processos finalizados pelo sistema por unidade de tempo. O escalonador deve tentar conseguir a maior taxa de serviço possível.
- **Tempo médio de resposta** - É o tempo decorrido desde a submissão de um job até a sua saída do sistema. Este tempo deve ser reduzido e sua variância mantida o mais baixo possível .
- **Consistência** - As respostas de um sistema não devem variar. Por exemplo, a execução de um mesmo processo deve ter tempo de resposta semelhante se executado em diversas horas do dia. Se as respostas do sistema variarem muito, os usuários nunca saberão o que esperar e o computador pode passar a ser considerado não confiável.
- **Utilização dos recursos** - Um outro objetivo é que um sistema operacional deve manter seus recursos ocupados. Os recursos estão lá para serem utilizados e não devem ficar ociosos. Uma métrica para quantificar este objetivo é a utilização que representa a fração do tempo que o recurso está ocupado.
- **Justiça** - Deve-se garantir que todos os processos do sistema tenham chances iguais de uso do processador.

Estes critérios de decisão são muitas vezes contraditórios. Por exemplo, garantir justiça não garante menor tempo médio de resposta. Além disso, o comportamento de cada processo é único e difícil de prever. Alguns consomem grande quantidade de

tempo fazendo entrada ou saída, enquanto outros ocupam o processador por horas a fio, se tiverem oportunidade. Para assegurar que nenhum processo execute por um tempo muito grande, todos os computadores modernos possuem um relógio interno, que periodicamente gera um sinal de interrupção, denominado interrupção de tempo. A freqüência destas interrupções depende do sistema operacional e do hardware utilizado, ficando em torno 50 a 100 ocorrências por segundo. Em muitos computadores o sistema operacional pode programar este temporizador para gerar interrupções em qualquer freqüência. A cada interrupção de tempo, o sistema operacional é posto para executar e decide se o processo corrente deve ou não continuar de posse do processador, ou se ele deve ser interrompido, cedendo lugar a outro processo.

O escalonamento pode ocorrer em três níveis: alto, intermediário e baixo [36]. Os escalonadores de alto nível verificam um conjunto de condições que devem ser satisfeitas antes de permitir que um processo entre no sistema, ou seja, é feito um controle de acesso para proteger a integridade e garantir a segurança do sistema. O escalonamento de nível intermediário determina quais processos podem realmente competir pelo processador. Por exemplo, processos que solicitam muita entrada e saída praticamente não competem pelo processador pois, na grande maioria de seu tempo, estão esperando uma operação de entrada e saída [37]. O escalonador intermediário atua sobre processos ativados que foram suspensos para entrada e saída ou para esperar a conclusão de outro processo. Quando não há memória principal para manter todos os processos, o escalonador intermediário faz a troca (*swap*) de processos entre a memória principal e a secundária. Às vezes, este escalonador suspende certos processos quando a demanda de recursos é excepcionalmente alta. O escalonador de baixo nível determina quais, dentre os processos prontos para execução, devem obter o controle do processador. Por vários motivos, o escalonador de baixo nível é o mais difícil de implementar. Ao contrário dos outros níveis, ele não é governado por eventos. Freqüentemente há muitos processos competindo pelo processador e o escalonador de baixo nível tem que determinar qual deles obterá acesso.

Há duas estratégias principais para os algoritmos de escalonamento de baixo nível: escalonamento preemptivo e não-preemptivo. No escalonamento não-preemptivo, o processo que assume o controle do processador o mantém até terminar. O sistema operacional não tomará o controle. A vantagem deste tipo de escalonamento é a simplicidade, uma vez que o controle só é passado de um processo para outro quando o primeiro processo termina. A desvantagem é a falta de resposta ao comportamento do

sistema. O processo que tem o controle do processador é servido amplamente e os demais têm que esperar.

No escalonamento preemptivo, um processo não pode ficar no controle do processador indefinidamente. Depois de um certo tempo, o sistema operacional pode decidir tomar-lhe o processador. Os processos prontos para executar devem usar o processador em turnos. Em geral, o sistema operacional pode tomar o processador de um processo em execução pelas seguintes razões:

- o processo termina (por erro ou por concluir sua tarefa) e entrega o processador;
- o processo gera uma solicitação pela qual ele deve esperar. Pode ser uma solicitação de entrada ou saída, uma falta de página ou uma falta de segmento. Neste caso, o escalonador de nível intermediário coloca o processo em estado de espera enquanto a solicitação de entrada ou saída é processada;
- o processo foi executado por um tempo longo. O sistema operacional pode decidir tirar o controle do processo em execução no momento. O contexto do processo será armazenado para que sua execução possa ser retomada posteriormente sem perda de informação. Assim, um dos outros processos prontos para executar será escalonado. O tempo máximo que o sistema permite que um processo use o processador sem ser interrompido é um parâmetro denominado *quantum.*

Os algoritmos não-preemptivos, apesar de serem de implementação mais simples, não são adequados para os sistemas de propósito geral, nos quais uma grande quantidade de usuários compete por recursos. Por outro lado, em sistemas dedicados, pode ser razoável que um processo-mestre crie um processo-filho e o deixe executar até terminar. A diferença destes sistemas para os de propósito geral é que, no primeiro caso, todos os processos estão sob controle de um único mestre, que sabe o que cada filho está fazendo e quanto tempo tal tarefa vai demorar.

3.2 Políticas de Escalonamento

Uma política de escalonamento para servidores Web baseada no tempo necessário para servir uma requisição foi proposta em [17]. O tempo de serviço foi definido com base no tamanho do arquivo requisitado. Os experimentos foram realizados em uma rede local e os resultados mostraram redução no tempo médio de resposta de até um quarto do tempo obtido pelo servidor Apache com política de escalonamento padrão. A implementação da política não implicou em alterações no *kernel* do sistema operacional. O escalonamento das requisições foi realizado no nível

de aplicação. Para isso, foi modificado um servidor Web de modo que fosse possível controlar a ordem na qual as chamadas de sistema *read*() e *write*() são feitas. Porém, neste caso, foi necessário limitar o número de *threads* na fila da rede e na fila de disco para que se obtivesse controle sobre o escalonamento realizado pelo sistema operacional. Esta limitação no número de *threads* causou uma diminuição da taxa de processamento do sistema. Para que esta limitação não ocorra, a política deve ser implementada no sistema operacional onde pode-se ter total controle sobre a ordem na qual as requisições são atendidas.

Os artigos [16, 18, 19] tiram proveito da grande variabilidade dos tempos de processamento observados na carga de serviço Web para propor a utilização da política de escalonamento *SRPT* em servidores Web. Esta política foi avaliada, tanto através de análise [18], quanto de simulação e implementação [16, 19]. Em todos os casos a política mostrou bom desempenho. Nesta política as requisições para arquivos pequenos são atendidas com maior prioridade. Isto reduz muito o tempo médio de resposta do sistema e, como os tamanhos de arquivo seguem distribuição de cauda pesada, o uso desta política não prejudica o processamento dos arquivos grandes.

Na carga de serviço de servidores Web, uma pequena fração dos maiores arquivos corresponde à maior parte da carga imposta ao sistema. Por exemplo, para uma carga de serviço que possa ser modelada por distribuição de Pareto[1] com α=1.1, 1% dos maiores arquivos representam mais de 50% de toda a carga imposta ao servidor [19]. Isso corresponde a dizer que os arquivos enquadrados na faixa de 1% maiores terão que disputar recursos com 50% da carga restante. O mesmo não se aplica a uma carga de serviço modelada por distribuição exponencial. Em uma carga de serviço modelada por este tipo de distribuição, 1% dos maiores arquivos representa menos que 5% da carga total. Isto que dizer que o arquivo enquadrado na faixa de 1% dos maiores terá que disputar recursos com os 95% restante da carga. Portanto, conclui-se que em uma carga de serviço modelada por distribuição de cauda pesada, os processos grandes são muito menos penalizados em comparação a cargas de serviço exponenciais. O artigo [19] apresenta resultados bastante convincentes neste sentido.

No artigo [09] também foi proposta uma política de escalonamento baseada no tamanho da requisição. Para evitar *starvation* das requisições grandes, o autor propôs que o cálculo da prioridade fosse baseado em três parâmetros, a saber: tamanho da requisição, tempo de chegada da requisição no sistema e uma constante para controlar o

balanceamento entre minimização do tempo de resposta e justiça. Bons resultados foram obtidos utilizando-se esta forma de cálculo. Porém, de acordo com os trabalhos [16, 17, 18, 19] e considerando as características da carga Web, a prioridade poderia ter sido definida apenas com base no tamanho da requisição. Muitas das premissas utilizadas no trabalho [09] foram também utilizadas nos trabalhos [16, 17, 18, 19] com exceção da necessidade de evitar *starvation* através do tempo de chegada da requisição.

A política de escalonamento padrão de servidores Web é a FIFO, na qual as requisições são admitidas no *pool* na ordem de chegada, e processadas pela política de escalonamento *processor sharing*. A seguir discutimos algumas das políticas de escalonamento mais conhecidas.

3.2.1 Escalonamento FIFO

Provavelmente a estratégia mais fácil de implementar seja a FIFO (*First In First Out*). Sua abordagem não-preemptiva é muito simples: dar o controle ao processo que está há mais tempo no sistema. Sua principal vantagem é a simplicidade. O sistema operacional faz o escalonamento apenas quando é absolutamente necessário. Não é indicado para sistemas em que os processos fazem muita entrada e saída, neste caso um escalonamento preemptivo é mais indicado. Contudo, em sistemas onde a maioria dos processos destina-se a processamento numérico, ou seja, muito processamento e pouca entrada e saída, o escalonamento FIFO pode apresentar bom desempenho [37].

3.2.2 Escalonamento Round Robin

Neste algoritmo, a cada processo atribui-se um *quantum* de tempo durante o qual ele poderá usar o processador. Se o processo ainda precisar executar depois de esgotado seu *quantum*, ele perde o processador, dando lugar a outro processo. Se o processo for bloqueado ou terminar antes de esgotado seu *quantum*, a comutação para um novo processo dar-se-á no exato momento do bloqueio ou do término do processamento [36].

As formas mais utilizadas para organizar a gerência da fila de processos são a FIFO e por prioridades, na qual os processos são ordenados por prioridades. O *round robin* (RR) é um algoritmo onde o tempo médio de resposta é independente da distribuição dos tempos de serviço. Nesta política, o tempo médio de resposta depende apenas da taxa de chegada de processos e do tempo médio de serviço.

---

[1] Distribuição de cauda pesada utilizada para modelar a distribuição dos tamanhos de arquivo na Web.

Um detalhe importante na implementação deste algoritmo é determinar o valor do *quantum*. Na troca de um processo para outro no processador gasta-se algum tempo em tarefas de gerenciamento tais como salvar e restaurar registradores e mapas de memória e atualizar as várias tabelas e listas. Deve-se avaliar o compromisso entre o tempo para a troca de contexto e o *quantum*, o que pode depender do ambiente no qual o escalonamento ocorre e do tipo de processo que será escalonado. Se o *quantum* for muito grande o RR tende a aproximar-se do escalonamento FIFO, se o *quantum* for muito pequeno o RR tende para uma variação chamada de *Processor Sharing* (PS). Neste algoritmo de escalonamento o *quantum* tende a zero e cada processo recebe uma fração do tempo do processador igual a 1/N, onde N é o número de processos no sistema [51].

3.2.3 Escalonamento com Filas Múltiplas

Em sistemas interativos, nos quais não há espaço na memória principal para todos os processos, é necessário fazer a troca de processos entre memória principal e secundária. A troca de contexto aumenta o tempo de execução. Nestas situações é interessante utilizar um *quantum* com valor alto. Contudo, um *quantum* alto pode gerar tempos de resposta muito grandes para sistemas interativos onde vários usuários competem por recursos [36]. Uma solução possível é dividir os processos em classes de prioridade. Os processos da classe mais prioritária executam com *quantum* 1. Aqueles que estiverem na classe seguinte executam com *quantum* 2. Na próxima, *quantum* 4, e assim por diante. Sempre que um processo esgota seu *quantum*, ele é posto na fila da próxima classe de prioridade, ou seja, se ele acabou de executar com *quantum* 2, vai para a fila com *quantum* 4.

Considere, como exemplo, um processo que precise de 100 unidades de tempo para ser concluído. Inicialmente ele receberá *quantum* 1, e depois de decorrido este tempo ele será suspenso. Da próxima vez ele receberá *quantum* 2; após isto será novamente suspenso. Nas execuções seguintes este processo receberá sucessivamente *quantum* de 4, 8, 16, 32 e 64, usando somente 37 dos últimos 64 alocados a ele, completando então o seu trabalho. Observa-se que só foram necessários 7 passos para que o processo termine seu processamento, em vez dos 100 que seriam necessários se estivesse usando o *round robin*.

3.2.4 Escalonamento com Prioridade

O escalonamento *round robin* assume de forma implícita que todos os processos são igualmente importantes. Dependendo da aplicação, pode ser interessante considerar outros fatores e atribuir prioridades aos processos. A idéia básica é a seguinte: a cada processo é associada uma prioridade, e o processo pronto com maior prioridade será aquele que vai executar primeiro [36]. Para evitar que processos com alta prioridade monopolizem o processador, o escalonador decrementa a prioridade do processo que está executando a cada interrupção de tempo. Se tal ação fizer com que a prioridade do processo corrente torne-se mais baixa que a do de mais alta prioridade da fila de prontos, deve ocorrer uma troca de contexto.

As prioridades podem ser atribuídas dinamicamente aos processos pelo sistema, de forma que um certo objetivo possa ser atingido. Por exemplo, alguns processos usam muito os dispositivos de entrada e saída, gastando a maior parte de seu tempo de processamento aguardando a finalização destas operações. Sempre que um processo com tal comportamento precisar do processador, este deve ser imediatamente entregue a ele, para deixá-lo iniciar a próxima operação de entrada e saída, que poderá prosseguir em paralelo com o novo processo ao qual o processador será alocado. Fazer com que processos com muita entrada e saída esperem um bom tempo pelo processador significa simplesmente que ele estará ocupando a memória por um longo espaço de tempo. Um algoritmo simples para dar mais prioridade aos processos com muita entrada ou saída é associar a cada processo do sistema uma prioridade de $1/f$, sendo f a fração do *quantum* que ele usou em sua última execução. Por exemplo, se o *quantum* for 100, um processo que usou só dois destes 100 vai ganhar uma prioridade de 50, enquanto que um outro que executou 50 do *quantum* antes de ser bloqueado, receberá prioridade igual a 2, e um processo que usou todo o *quantum* terá prioridade de 1.

Algumas vezes pode ser conveniente agrupar processos em classes de prioridades e usar o escalonamento com prioridade entre as classes e o *round robin* dentro de cada classe. A figura 2 mostra um sistema com quatro classes de prioridades. O algoritmo de escalonamento fica, então, assim: enquanto houver processos prontos na classe de prioridade 4, tais processos vão sendo escolhidos para executar segundo a política *round robin*, não havendo nenhuma escolha de processos pertencentes às classes de prioridade mais baixa. Se a fila da classe de prioridade 4 estiver vazia, os processos da classe 3 vão executar escolhidos por *round robin*. Se as filas das classes 4 e 3 estiverem vazias, a escolha recai sobre os processos da classe 2, e assim por diante.

No exemplo da figura 2 o processo P8 somente será executado depois que todos os processos das filas 3 e 4 forem executados. Se as prioridades não estiverem bem sintonizadas, pode ocorrer uma situação de *starvation,* onde processos nas classes mais baixas nunca tenham oportunidade de executar. Por isso pode ser necessário algum critério dinâmico, tal como o tempo na fila, para evitar esta situação.

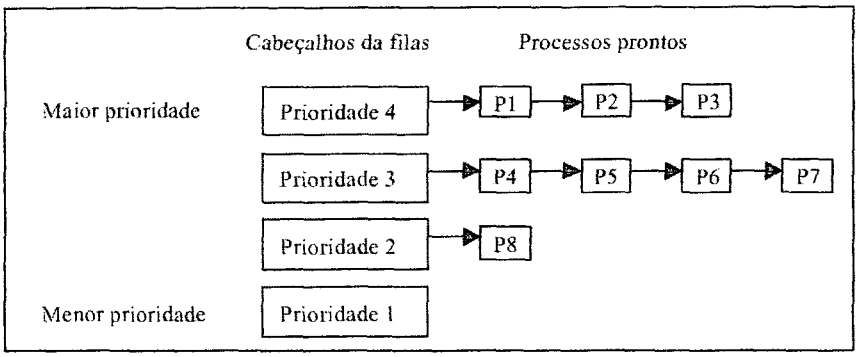

Figura 2 : Escalonamento baseado em quatro níveis de prioridade.

3.2.5 Menor Processo Primeiro (Shortest Job First)

Este algoritmo é apropriado para sistemas nos quais o tempo de processamento de cada processo é conhecido com antecedência, como é o caso do processamento de requisições HTTP estáticas. Nesta política, quando vários processos igualmente importantes estiverem esperando sua vez numa fila, o escalonador usa a política de alocar o processador ao menor dos processos da fila. Para comprovar a vantagem desta política em relação a FIFO vamos fazer uma demonstração prática. Na figura 3a há quatro processos, A, B, C e D, que precisam de 7, 5, 1, e 3 segundos respectivamente para realizar seu processamento. Efetuando o processamento utilizando a metodologia FIFO da esquerda para a direita, o tempo de permanência no sistema para A é 7 segundos, para B 12, para C 13 e para D 16 segundos, dando uma média de 12 segundos por processo.

Figura 3 : Escalonamento FIFO e PS *versus* SJF.

Ao executar os processos usando a política SJF também da esquerda para a direita, como mostrado na figura 3b, os tempos de permanência serão respectivamente de 1, 4, 9 e 16 segundos, com uma média de 7,5 segundos. O modelo matemático desta

política pode ser assim descrito: supondo quatro processos, com tempos de processamento iguais a *a, b, c, d*, respectivamente, o primeiro processo permanece no sistema um tempo *a*, o segundo um tempo igual a *a* + *b*, o terceiro um tempo igual a *a*+*b*+*c* e assim por diante. Desta forma, o tempo médio de permanência é dado por (4*a* + 3*b* + 2*c* + 1*d*)/4. Para calcular este tempo médio deve-se multiplicar o tempo de *a* por quatro uma vez que o tempo de *a* fará parte do tempo de permanência dos quatro processos; o tempo de *b* deve ser multiplicado por três pois ajuda a compor o tempo de permanência dos três últimos processos, e assim sucessivamente. Fica claro que *a* contribui mais para o tempo médio do que os outros tempos, de forma que ele deve ser o tempo do menor processo, sendo *b* o tempo do segundo, depois *c*, e finalmente *d* deve corresponder ao tempo gasto pelo processo mais demorado.

A política SJF apresenta menor tempo médio de permanência inclusive quando comparada com PS. Utilizando-se os mesmos dados apresentados na figura 3, observamos que quando aplicada a política PS no exemplo da figura 3a, temos um tempo médio de 10,75. Quando os processos da figura 3b são executados pela política PS temos um tempo médio de permanência de 9,5. Deve ressaltar que o exemplo da figura 3b é o melhor caso para a política PS e mesmo assim apresenta um tempo superior ao da política SJF, que foi de 7,5.

A versão preemptiva deste escalonamento é denominada *SRPT (Shortest Remaining Processing Time)*. Dois argumentos são alegados como desvantagens importantes desta política de escalonamento. Primeiro, a dificuldade em predizer quanto tempo gastará um processo. Segundo, a possibilidade de haver o adiamento indefinido ou *starvation* de um processo grande. Enquanto houver processos mais rápidos chegando ao sistema, os processos mais demorados não serão executados. Contudo, em servidores Web estas desvantagens são praticamente anuladas. A primeira dificuldade é eliminada uma vez que o servidor Web, a partir do recebimento do pedido HTTP, pode determinar com antecedência qual será o custo do processo que atenderá a requisição. A segunda desvantagem é descartada devido à distribuição dos tamanhos de arquivos solicitados em requisições HTTP estáticas, que é uma distribuição de cauda pesada. Diversos trabalhos [16, 18, 19] comprovaram que neste tipo de carga de serviço não ocorre *starvation* dos processos grandes.

3.2.6 Escalonamento Garantido

Uma forma completamente diferente de tratar a questão do escalonamento é fazer certas promessas ao usuário à respeito do desempenho, e cumprí-las de alguma forma. Uma promessa bem realista e muito fácil de cumprir é a de que se houver ***n*** usuários ativos na rede, cada um vai receber em torno de $1/n$ da capacidade de processamento do processador [36].

Para cumprir esta promessa, o sistema deve controlar o total do tempo do processador que um usuário usou para todos os seus processos desde o momento em que tal usuário tornou-se ativo. Ele então calcula o tempo que cada usuário deveria merecer, simplesmente dividindo o tempo decorrido desde sua ativação por ***n***. Como o tempo que cada usuário gastou até o momento é conhecido, é fácil calcular a razão entre o tempo realmente concedido ao usuário e o tempo prometido. Uma razão de 0,5 significa que o processo teve alocado somente metade do tempo prometido. Já no caso da razão ser 2,0 significa que o processo já gastou o dobro do tempo de processador que lhe foi prometido. O processo a executar será sempre o que possuir razão mais baixa.

A literatura de sistemas operacionais fala em um número bastante grande de outros algoritmos de escalonamento de processos [25, 35, 36, 37]. O trabalho básico do escalonador é determinar qual processo vai executar a seguir, levando-se em conta vários fatores como o tempo de resposta, a eficiência, justiça entre outros aspectos particulares de cada aplicação.

4. A POLÍTICA *FASTEST CONNECTION FIRST* (FCF)

Este capítulo tem por objetivo fazer a apresentação da nova política de escalonamento incluindo aspectos de sua implementação em um sistema real. Nele é descrito como a prioridade é definida e quais os locais do sistema onde a política deve ser aplicada. Também são feitas observações sobre como a política coopera para melhorar a interação entre servidor Web e Internet.

4.1 Descrição da Política FCF

Um servidor Web geralmente atende várias requisições concorrentemente. Alguns servidores implementam esta multiprogramação criando um novo processo ou *thread* para cada nova conexão. Dependendo da implementação, o custo para criar um novo processo pode ser alto, gerando um *overhead* inaceitável. Outros servidores, na tentativa de minimizar o *overhead*, implementam um mecanismo conhecido como *pool* de processos, onde um determinado número de processos é criado durante a inicialização do servidor [08]. Seja qual for o caso, o servidor não faz uso de prioridades para o tratamento das requisições [01, 09].

O servidor Web Apache, por exemplo, cria um *pool* de processos e trata as requisições de acordo com a ordem de chegada (FIFO), não importando seu tipo ou tamanho. As requisições admitidas no *pool* são atendidas, concorrentemente, pela política PS. Assim, a escolha do processo que pode utilizar um recurso do sistema (processador, disco ou interface de rede) é feita pela ordem de chegada da requisição.

A política FCF muda a ordem na qual as requisições ou processos recebem recursos do sistema. O próximo processo a receber recursos será aquele que estiver atendendo à requisição de maior prioridade. Esta prioridade será definida em função de dois parâmetros: menor quantidade de bytes a processar e maior taxa estimada de transmissão. A idéia é dar prioridade ao processo cuja conexão correspondente pode ser encerrada mais rapidamente, isto é, à menor requisição feita através da conexão com a maior taxa de transmissão.

O primeiro parâmetro é utilizado para dar prioridade ao menor processo. O segundo parâmetro visa dar prioridade à requisição que pode ser transmitida no menor tempo possível. A taxa de transferência depende das condições de conectividade do

usuário e das variações dinâmicas da Internet. Na seção 5.4 será descrito um modelo para simulação das condições de conectividade da Internet.

A política FCF opera da seguinte forma. Primeiramente busca-se na fila de requisições aquela com o menor número de bytes a processar. A seguir, outra pesquisa é realizada na mesma fila, buscando-se a requisição em processamento que tem a maior taxa de transmissão estimada. Esta nova pesquisa, porém, é restrita às requisições que possuírem um número de bytes a processar menor ou igual ao produto de uma constante (*beta*) maior que um, pelo número de bytes encontrado na primeira pesquisa. *Beta* define a abrangência da faixa de pesquisa. Ao variar o valor de *beta* damos maior ou menor peso para a taxa de transmissão ou para o tamanho do processo. O cálculo da taxa de transmissão é feito dividindo-se o tempo estimado para transferir o arquivo pelo tamanho do arquivo. Quanto maior for esta taxa, maior será a prioridade da requisição.

O objetivo da política é considerar as condições de conectividade do usuário e todos os aspectos inerentes à transmissão de dados pela Internet. Conexões mais velozes tendem a ter RTT menores e janelas de receptor e congestionamento maiores [14]. Consequentemente, as requisições feitas através destas conexões, se atendidas com maior prioridade no servidor, podem ser concluídas mais rapidamente, liberando os recursos do servidor para o atendimento de novas requisições.

Requisições Web são servidas através de uma conexão TCP. Neste protocolo, cada pacote ou conjunto de pacotes de dados, enviado pelo remetente, deve ser confirmado pelo destinatário. Devido ao controle de congestionamento realizado pelo TCP (*slow start*), apenas um número limitado de pacotes pode ser enviado a cada vez, devendo os demais aguardar confirmação da chegada dos pacotes já enviados. As confirmações das conexões velozes chegam mais rapidamente e, consequentemente, novos pacotes podem ser enviados. A política FCF visa garantir que os *buffers* de *socket* das conexões velozes sempre possuam dados para enviar quando as confirmações chegarem. Ou seja, os pacotes já estarão disponíveis para o envio quando puderem ser enviados. A política FCF procura tratar a requisição no servidor de acordo com o nível de conectividade medido dinamicamente para a respectiva conexão. Se a conexão é rápida, a requisição também será tratada rapidamente no servidor. Portanto, o ponto fundamental desta política é garantir que os *buffers* das conexões mais rápidas recebam mais rapidamente os dados do sistema de arquivos. Desta forma, quando um ACK de uma conexão rápida chega, não haverá necessidade de ler os dados do sistema de arquivos pois estes já estarão no *buffer*. Em situações em que o servidor Web opera

com *uma carga moderada* ou baixa, quando uma confirmação chega, os *buffers* das conexões normalmente já têm dados a transmitir [26] independente da política de escalonamento aplicada. Contudo, quando o servidor opera sobre uma carga elevada, dar prioridade às conexões rápidas trará mais garantias de que seus *buffers* tenham dados a transferir quando da chegada de confirmações (ACK). Além disso, com carga elevada tornam-se maiores as filas de mensagens de controle da saída do TCP, de datagramas IP e de pacotes na interface de rede. Desta forma, a política de escalonamento poderia ser aplicada também nestas filas, de modo a garantir que os pacotes das conexões mais velozes cheguem mais rapidamente à interface de rede. O trabalho realizado em [19] aplicou prioridade baseada no tamanho do arquivo apenas na fila de pacotes da interface de rede obtendo bons resultados.

Outra contribuição importante desta política de escalonamento é a de proporcionar redução do número de requisições pendentes. Como poderá ser observado nos resultados, esta redução acontece em decorrência do atendimento prioritário dado às requisições que são transmitidas mais rapidamente pela Internet. A redução do número de requisições pendentes proporciona uma conseqüente redução das trocas de contexto, as quais, como observado no trabalho [32], podem prejudicar o desempenho de servidores Web. A comprovação de que o excesso deste tipo de requisição prejudica o desempenho de servidores Web foi apresentada no trabalho [34]. Os resultados deste trabalho demonstraram que o servidor Web Apache, trabalhando sobre uma rede com RTT médio de 200 milissegundos, pode ter seu desempenho reduzido em mais de 50% comparativamente com o trabalho sobre uma rede local. A redução no desempenho foi provocada justamente pelo aumento das trocas de contexto e pela necessidade de mais memória para gerenciar um maior número de requisições concorrentes.

4.2 Considerações sobre a Implementação da Política FCF

A política FCF foi avaliada através de simulação na qual foi considerada a interação entre o servidor e a Internet. A metodologia de avaliação é apresentada no capítulo 5 e os resultados obtidos no capítulo 6. Uma parte do modelo da simulação é a estimativa da taxa de transmissão. Em uma situação real, a estimativa da taxa de transmissão (*bandwidth* da conexão) deve ser feita utilizando-se parâmetros que possam informar a condição de conectividade do usuário, por exemplo, o tamanho das janelas de congestionamento e do receptor e a estimativa do RTT médio.

Na implementação real desta política deve se considerada a forma e os motivos pelos quais os dados das requisições HTTP são enviados através do TCP em um servidor Web. O envio de dados pelo TCP ocorre em três situações. Primeiro, o servidor Web, depois de fazer o *parser* do cabeçalho da requisição, faz uma chamada à primitiva *write* do *socket* e os dados são transferidos do sistema de arquivos ou *cache* de disco para o *buffer* da conexão. Neste ponto uma mensagem de controle é inserida na fila de saída do TCP indicando que dados podem ser enviados. Deve-se observar, contudo, que o TCP enviará a quantidade de dados permitida pelas janelas de congestionamento e do receptor. O envio de dados é também necessário em caso de *timeout*, quando é feita a retransmissão. Neste caso uma mensagem de controle também é colocada na fila de saída do TCP. Na terceira situação, um segmento de confirmação (ACK) é recebido pela interface de rede e mais dados do *buffer* da conexão podem ser enviados. Novamente a quantidade de dados enviada dependerá das janelas de congestionamento e do receptor.

Uma vez que mensagens de controle são adicionadas na fila de saída do TCP, o processo que executa o TCP pode enviar segmentos para a fila do IP. Neste ponto, o processo do IP envia datagramas para a fila da interface de rede. Finalmente, o processo que controla a interface de rede pode transmitir, em segundo plano, os pacotes que estão em sua fila.

Considerando o procedimento de transferência de dados através do TCP, a política FCF deveria ser aplicada nos seguintes locais:

- fila de mensagens de controle da saída do TCP;
- fila de datagramas esperando processamento do IP;
- fila de pacotes da interface de rede;
- fila dos processos para execução da primitiva *write*.

Alterar a ordem de processamento da fila de saída do TCP implica, na prática, em alterar a implementação do TCP. Para a fila de datagramas IP seria necessário alterar a implementação do IP. Alterar a ordem de processamento das primitivas *write* implica em mudar o sistema operacional estabelecendo prioridades no disco e CPU. É importante ressaltar que todas as alterações devem ser feitas no servidor, não implicando portanto, alterar software do usuário ou da rede (p. ex. roteadores).

A ordem de processamento na interface de rede poderia ser alterada, no ambiente Linux, utilizado-se o *Patch DiffServ* [21]. Com este *patch* é possível implementar 16 filas de prioridade (0 a 15). Este recurso do Linux foi originalmente

criado para tratar o byte de qualidade de serviço do pacote IP. Contudo, ele pode ser utilizado para passar informações de prioridade baseando-se em outros parâmetros [19]. As informações necessárias para definir as prioridades podem ser facilmente obtidas do TCB (*Transmission Control Block*), campos *tcb_cwnd, tcb_swindow* e *tcb_srt* [27]. O tamanho do arquivo a ser transferido pode ser obtido do sistema de arquivos tão logo a conexão tenha sido estabelecida e o pedido HTTP tenha sido recebido. De posse destas informações é possível configurar a prioridade da conexão utilizando-se da chamada de sistema *setsockopt()*. A descrição exata de como configurar a prioridade da conexão é apresentada em [19]. A descrição dos campos *tcb_cwnd, tcb_swindow* e *tcb_srt* do TCB e como acessá-los é dada em [27].

5. MODELO DE SIMULAÇÃO PARA A POLÍTICA FCF

Neste capítulo é apresentada a descrição dos modelos utilizados para simular os componentes do ambiente onde a política FCF foi avaliada. O ambiente de simulação compreende a modelagem do servidor Web, da carga de serviço e da Internet. No servidor foram simulados os recursos de CPU, disco e interface de rede. A carga foi simulada utilizado-se os conceitos e fórmulas do SURGE. A Internet foi simulada através de distribuições de *bandwidth*, RTT e perdas de pacotes e da modelagem dos protocolos TCP e HTTP/1.1.

5.1 Descrição do Modelo

O desempenho de um servidor Web pode ser avaliado de diversas formas, incluindo simulação, modelos analíticos e experimentação. Para que seja possível fazer experimentação é necessário que o sistema a ser avaliado já tenha sido implementado, ou que um sistema existente possa ser alterado de modo a reproduzir o sistema a ser avaliado. No caso específico da avaliação da política FCF, o uso de experimentação torna-se complexo e demorado por necessitar que sejam alterados tanto o sistema operacional quanto as implementações do TCP e IP. O problema de avaliar FCF e outras políticas, através de experimentação em um sistema real é a dificuldade de controlar a Internet. A eficiência da FCF está diretamente ligada às condições de conectividade da Internet. Em um ambiente real é muito difícil reproduzir duas ou mais vezes as mesmas condições de conectividade para que as políticas possam ser comparadas. Além disso, a implementação, embora possível, levaria um tempo muito maior do que o disponível para este trabalho.

Modelos analíticos são baseados em um conjunto de fórmulas e algoritmos computacionais usados para avaliar o desempenho do sistema proposto. A simulação, por sua vez, é realizada através de programas que imitam o comportamento do sistema [50]. Ela é uma técnica complementar dos modelos analíticos [41]. Através dela é possível verificar a correção da análise e confirmar as suposições sobre o modelo. Além disso, permite avaliar o modelo pela exploração de cenários complicados que muitas vezes são difíceis ou até impossíveis de serem validados apenas pela modelagem. Por este motivo a técnica escolhida para avaliar as políticas foi a simulação.

Seja qual for a metodologia utilizada para avaliar um servidor Web, ela deve modelar os três principais componentes deste ambiente, a saber: os usuários que representam a carga de serviço, o servidor Web e a Internet. A figura 4 apresenta este ambiente. No decorrer deste capítulo será descrito como foram simulados estes componentes do ambiente de simulação.

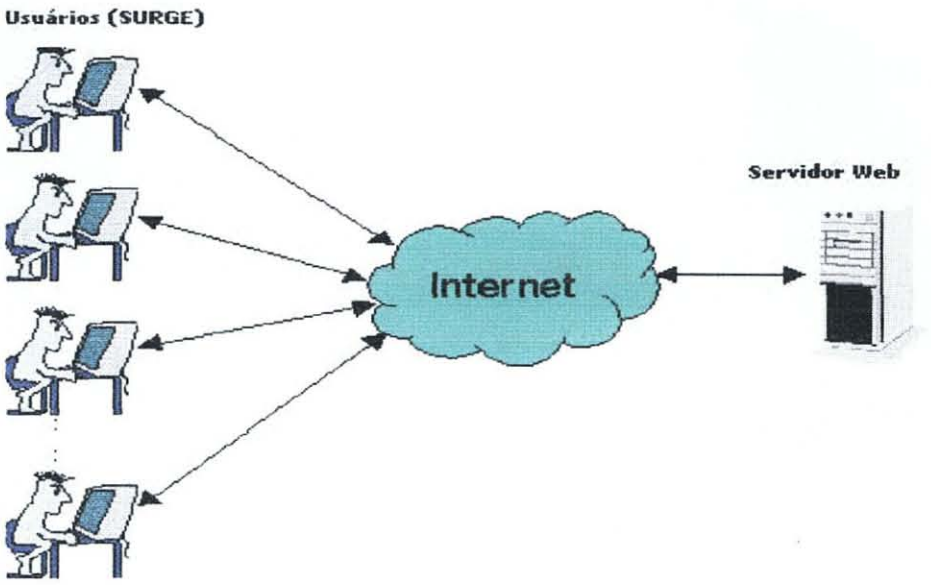

Figura 4 : Representação do ambiente de simulação.

O programa de simulação foi inicialmente implementado na linguagem Pascal versão 7.0. Posteriormente, devido à falta de recursos neste ambiente de programação, o programa foi migrado para a versão 5.0 da linguagem Borland Delphi Enterprise. O programa é composto por duas bibliotecas (*Units*). A primeira, com aproximadamente 1200 linhas, contém o código da simulação propriamente dita. A segunda, com cerca de 600 linhas, destina-se a constantes, definição de tipos estruturados e implementações de funções e estruturas de dados utilizadas, como filas e listas encadeadas.

5.2 Simulação do Servidor

Um servidor Web é um programa que aceita conexões com o objetivo de servir requisições de acesso às páginas Web armazenadas no seu sistema de arquivos. Estas informações podem ser obtidas de forma estática ou dinâmica. O objetivo desta simulação é avaliar o servidor Web no processamento de requisições de conteúdo estático. O atendimento deste tipo de requisição utiliza majoritariamente três recursos do sistema: CPU, disco e interface de rede. O processamento das requisições estáticas no servidor pode ser descrito da seguinte forma: estabelecimento da conexão TCP,

recebimento do pedido HTTP, processamento do cabeçalho da requisição, leitura do arquivo do disco ou do *cache*, empacotamento e envio dos dados para o usuário, fechamento da conexão e registro no arquivo de *log*. De acordo com os trabalhos [05, 09, 17], dentro desta seqüência de ações a demanda de serviço para uma requisição pode ser dividida em três partes, a saber:

- tempo de CPU : representa o tempo gasto pela CPU para fazer o *parser* da requisição, empacotamento e desempacotamento dos segmentos TCP e datagramas IP, controle da transferência dos dados do disco para o *buffer* de *socket* e deste para a interface de rede, entre outros aspectos da manipulação de uma requisição HTTP;
- tempo de leitura : é o tempo gasto pelo disco para transferir o arquivo solicitado para o *buffer* do *socket*;
- tempo de processamento na interface de rede : tempo gasto para transferir para o *buffer* da interface de rede os dados solicitados.

Seguindo esta divisão, cada uma das três partes da demanda de serviço de uma requisição pode ser associada a um dos três recursos do sistema, CPU, disco e interface de rede. Assim, o servidor Web foi simulado utilizando-se três filas, uma para cada recurso citado. Os trabalhos [09, 17, 18, 19] também utilizam modelagem semelhante. A figura 5 ilustra o fluxo de execução das requisições no servidor simulado. Nesta figura observamos que as novas requisições são colocadas inicialmente na fila de CPU. Somente após realizado o processamento na CPU a requisição passa para a fila de disco para que os dados sejam transferidos deste para o *buffer* de *socket*. Depois disso, a requisição passa para a fila da interface de rede aqui denominada fila de rede.

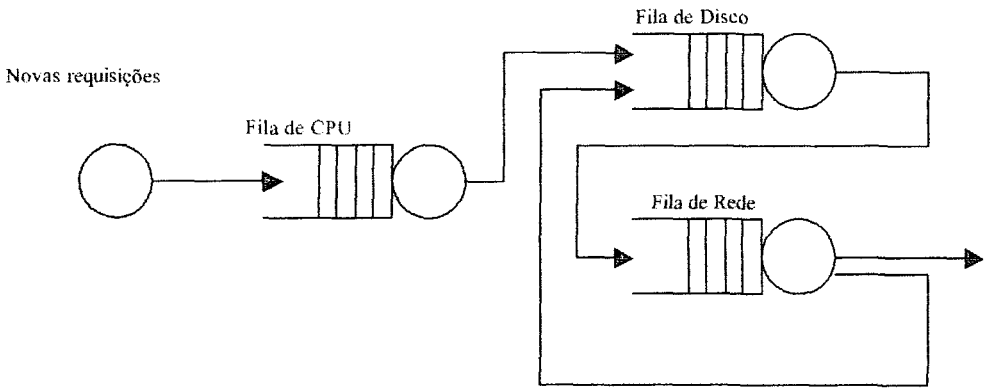

Figura 5 : Representação das filas de um servidor Web.

As entidades armazenadas nas filas correspondem a requisições HTTP. Por considerar que nos estágios de recebimento e *parser* da requisição ainda não estão disponíveis as informações para definir a prioridade, a fila de CPU é gerenciada pelo modelo FIFO e suas requisições são processadas de acordo com a política PS. Este modelo é considerado uma abstração da política padrão utilizada em servidores Web [09, 24].

O processamento da fila de disco consiste em transferir dados do arquivo requisitado, passando estes dados do sistema de arquivos para o *buffer* do *socket*. Os dados são lidos do disco em blocos de 16 Kbytes (*block mode*). Depois de ler um bloco de dados, o descritor da conexão é transferido para a fila de rede. A fila da interface de rede terá o custo de transferir os dados do *buffer* do *socket* para o *buffer* da interface de rede. Neste ponto os pacotes podem ser enviados pela rede. Uma abordagem semelhante a esta é utilizada no trabalho [26]. Quando a requisição tiver todos os seus dados transferidos para o *buffer* da interface de rede ela é retirada da fila. Contudo, caso existam dados remanescentes, a requisição deve voltar para a fila de disco aguardando que este envie novos dados para o *buffer* do *socket*.

A requisição somente estará concluída quando tiver cumprido a soma dos custos das filas de CPU, disco e rede, considerando que uma requisição não pode ser processada simultaneamente nos três recursos do sistema (o processamento é concorrente). A figura 6 mostra a linha de tempo para a simulação do processamento das requisições HTTP. Nela observamos que a requisição fica alternando entre as filas de disco e rede até ser concluída. Apenas o envio de dados pela Internet ocorre simultaneamente ao processamento da requisição.

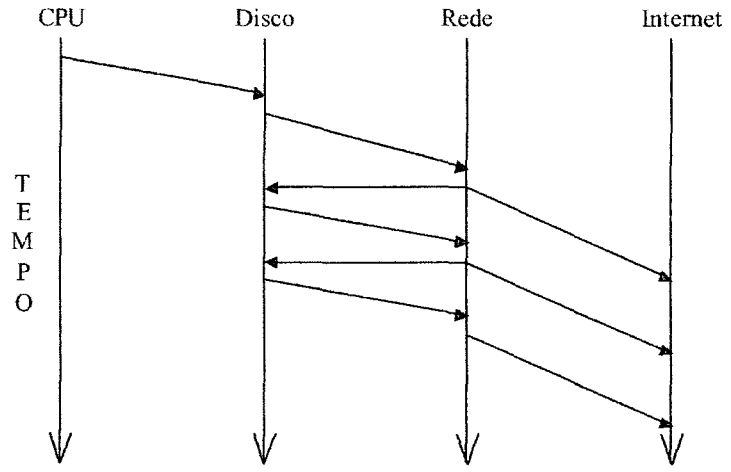

Figura 6 : Diagrama de tempo para o processamento de requisições HTTP.

Diversos trabalhos [04, 09, 17, 18, 19] têm demonstrado que, para requisições estáticas, o custo de processamento é proporcional ao tamanho da requisição. Portanto, o custo de cada requisição foi calculado com base no tamanho do arquivo requisitado. Contudo, como já foi demonstrado, este custo foi distribuído entre os três recursos do sistema, a saber: CPU, disco e interface de rede. O custo de CPU definido para a simulação foi de 0,2 milissegundos por Kbyte processado. Este valor foi calculado a partir do trabalho [04] que fornece dados sobre o tempo de processamento para requisições Web. Pelo fato do processador utilizado naquele trabalho estar bastante desatualizado se comparado com os padrões atuais, o valor usado nesta simulação é muitas vezes menor. Possivelmente este valor estará desatualizado em breve devido ao contínuo aumento da capacidade dos processadores. Contudo, o fato do processador ter menor ou maior capacidade não afetará os resultados da comparação, considerando que as políticas são avaliadas sobre as mesmas condições de carga e processamento. Ao custo de CPU, que é proporcional ao número de bytes, foi acrescido um valor constante referente ao tempo de *parser* e ao tempo de registro (*logging*). Os valores são respectivamente 1 e 1,5 milissegundos.

Para calcular a demanda de disco foi considerado um disco com taxa de transferência de 80 Mbytes por segundo e velocidade de rotação de 7200 RPM. O tempo médio de latência utilizado foi de 4,17 milissegundos. Este valor foi obtido a partir da fórmula dada em [50], a saber:

Tempo médio de latência = (60/7200*1000)/2.

A divisão por dois é utilizada porque o tempo de latência para acessar um bloco qualquer do disco deve estar entre zero e o tempo para o giro completo do disco. O tempo de *seek* para a leitura de um bloco foi de 3 milissegundos e o *blockmode* do disco utilizado foi de 16384 bytes.

Portanto, a demanda de disco para ler um bloco foi calculada a partir da seguinte fórmula: DD=4,17+3+(16384/80000000*1000) = 7,37 milissegundos.

O custo de disco para leitura de um arquivo foi calculado a partir da fórmula:

Custo de disco = ([BLD/16384]) *DD, onde BLD é o número de bytes lidos do disco. A divisão de BLD/16384 é sempre arredondada para a próxima unidade inteira.

Deve-se observar que *cache* de disco não foi modelado nesta simulação. Também não foi considerada a leitura consecutiva de blocos do disco sem que houvesse necessidade de novos tempos de *seek*. Todavia, para minimizar este fato, o tempo de

seek foi mantido um pouco aquém dos valores observados em equipamentos comercializados no mercado atual.

A demanda de serviço para a interface de rede foi calculada levando-se em consideração um dispositivo para rede local de 10 Mbps. Portanto, para calcular a demanda de rede foi utilizado o valor de 0,819 milissegundos por Kbyte. Fórmula:
Custo de rede por Kbyte = (1000/(10000000/8/1024)) = 0,819 ms.

5.3 Simulação da Carga

O segundo componente do ambiente de simulação é a carga de serviço imposta ao servidor Web. Esta carga corresponde ao conjunto de requisições HTTP recebidas durante um período de tempo. Para que seja possível avaliar um servidor Web é necessário gerar com precisão a carga de serviço. Há duas formas normalmente utilizadas para gerá-la, a saber: baseada em arquivos de *log* (carga dirigida por *trace*) e baseada em modelos estatísticos (carga estocástica). A primeira forma faz a reprodução de todas as requisições armazenadas em arquivos de *log* coletados de servidores Web reais. Este tipo de abordagem é fácil de implementar e garante uma representação fiel da carga imposta por usuários Web. No entanto, a utilização de cargas reais apresenta duas desvantagens: não pode ser parametrizada e nem é escalável. O fato de não ser possível a parametrização impede que sejam alteradas características da carga de modo a avaliar o sistema sob diferentes aspectos. Não havendo escalabilidade não é possível reduzir ou aumentar a intensidade da carga, o que permitiria avaliar o servidor sob outras demandas de serviço. Alterações nos parâmetros da carga, como por exemplo, diminuir o tempo entre chegadas, podem fazer com que a carga perca sua identidade, não mais representando com fidelidade o conjunto de usuários [49].

A segunda forma utilizada para geração da carga permite que sejam alteradas as características e a intensidade da carga. Os geradores de carga que utilizam modelos estatísticos fazem requisições tão rápido quanto o servidor pode atender, a partir de um pequeno conjunto de arquivos do servidor Web. Este tipo de geração de carga, embora seja escalável e parametrizável, pode não representar fielmente as características reais das cargas de serviço observadas na Web. É importante que o gerador seja parametrizável, mas ele deve também representar com precisão as características reais de uma carga Web.

Nos trabalhos [03, 23] é apresentado um gerador que pode ser configurado e ao mesmo tempo reproduz todas as características que podem ser observadas em cargas

Web. Este gerador é chamado de SURGE (*Scalable URL Reference Generator*). Todos os conceitos e fórmulas deste gerador foram utilizados na avaliação da política FCF. O SURGE é um *benchmark* utilizado para medir o desempenho de servidores Web através de um fluxo contínuo de requisições que simula o acesso de centenas ou milhares de usuários. Estes acessos reproduzem as principais características de uma carga real de serviço imposta a um servidor Web. Dentre estas características estão a distribuição de cauda pesada dos tamanhos de arquivo, a localidade temporal, a popularidade dos arquivos, o número de arquivos por página, o *think time* e o número de usuários simultâneos. Cada uma destas características será descrita no decorrer desta seção.

5.3.1 Distribuição dos Tamanhos de Arquivo

A carga de serviço imposta a servidores Web é bastante desigual, devido principalmente à distribuição dos tamanhos de arquivos. O servidor Web pode ter que tratar arquivos multimídia enormes e, também, arquivos contendo algumas dezenas de bytes [31]. A caracterização dos tamanhos dos arquivos na Web pode ser modelada por uma distribuição de cauda pesada. Este tipo de distribuição tem características diferentes das distribuições de probabilidade comumente utilizadas na modelagem e análise de sistemas distribuídos. Uma variável aleatória x segue a distribuição de cauda pesada se:

$$\overline{F}(x) = P[X > x] \sim x^{-\alpha}, \quad para\ x \to \infty,\ 0 < \alpha < 2.$$

onde $\overline{F}$ é o complemento de F que, por sua vez, é a função de distribuição cumulativa. Variáveis que seguem essa distribuição têm variância infinita e, se $\alpha < 1$, têm média infinita. Em termos práticos, isso significa que valores bem grandes para x são possíveis com probabilidade não desprezível. Uma distribuição típica desta classe é a de Pareto [13], cuja função de densidade de probabilidade é dada por:

$$f(x) = \alpha k^{\alpha} x^{-\alpha-1}, \quad \alpha, k > 0, \quad x \geq k,$$

Uma explicação para esta variação nos tamanhos dos arquivos na Web talvez seja a ocorrência de grandes arquivos de vídeo, código binário e imagens que contribuem para aumentar a cauda da distribuição. Como conseqüência disto, uma

pequena porcentagem de grandes arquivos representa a maior parte do bytes a serem processados pelo servidor e transferidos pela Internet.

Apesar da probabilidade não desprezível de requisições a arquivos muito grandes, alguns estudos [19, 28, 31] mostram que mais de 90% das requisições são para arquivos pequenos (menores que 10 Kbytes) e que, por outro lado, estas requisições representam pequena fração dos bytes transferidos na Web.

5.3.2 Distribuição da Freqüência de Acesso aos Arquivos

Outra característica importante da carga de serviço Web é a freqüência de acesso aos arquivos, também denominada popularidade. Esta característica da carga Web pode ser modelada pela lei de *Zipf* [13, 23]. Esta lei foi originalmente aplicada para definir a freqüência em que as palavras ocorrem em um texto. Se as palavras de um texto forem ordenadas pela sua freqüência, sendo P o número de ocorrências de uma dada palavra, então pela Lei de *Zipf* $P = kr^{-1}$ [29], onde k é uma constante positiva qualquer e r é a posição da palavra na seqüência ordenada. Ou seja, o número de ocorrências da primeira palavra da lista é duas vezes maior que o número de ocorrências da segunda palavra da lista. Da mesma forma, na Web, se considerarmos uma lista dos arquivos mais requisitados, ordenada pelo número de requisições, o arquivo mais popular é duas vezes mais requisitado do que seu sucessor na lista de arquivos mais acessados. A distribuição dos acessos aos arquivos tem forte efeito sobre o comportamento dos *caches* uma vez que os arquivos mais populares tendem a permanecer nos *caches*. No modelo da carga, a popularidade deve ser combinada com a distribuição dos tamanhos dos arquivos para gerar uma seqüência de requisições com tamanhos que seguem distribuição de cauda pesada.

5.3.3 Distribuição do Número de Arquivos por Página

Uma página Web pode conter inúmeros objetos embutidos (referenciados), os quais podem ser arquivos de imagem, som, vídeo, código binário, entre outros. O número de objetos de uma página tem grande influência na caracterização da carga de serviço. A carga decorrente da requisição de um URL que contém um arquivo HTML com referências para cinqüenta outros arquivos é muito diferente da submissão de um URL de um arquivo que não possui referências internas. No primeiro caso, tão logo o *browser* do usuário tenha recebido a página HTML principal, cinqüenta novas requisições serão geradas para que o servidor as atenda. Em [03] o número médio de

objetos por página HTML foi extraído através da análise dos arquivos de *log* gerados por servidores Web. No arquivo de *log* foram pesquisadas seqüências de requisições de um mesmo usuário que eram solicitadas em intervalos menores que um segundo. Foi considerado que um mesmo usuário não submete mais de um URL em um intervalo de tempo inferior a um segundo, a não ser que seja uma URL referenciada por outra página. Depois de analisar diversos arquivos de *log* foi concluído que o número de objetos referenciados em páginas Web pode ser caracterizado através de uma distribuição de Pareto limitada com α=2,43 e k=1. O número máximo de objetos é de 150.

5.3.4 Distribuição da Localidade Temporal

Localidade temporal é outra característica da carga Web que deve ser modelada. Ela refere-se à probabilidade de acesso de um mesmo arquivo dentro de um intervalo de tempo. Esta característica da carga Web tem influência no desempenho do sistema de *cache*.

A localidade temporal pode ser medida usando-se o conceito de profundidade de pilha [35]. Quando um arquivo é inicialmente requisitado, ele é colocado no topo de uma pilha, empurrando os demais arquivos da pilha uma posição para baixo. Quando o arquivo é novamente requisitado, sua posição corrente da pilha é registrada e ele é movido de volta para o topo da pilha. Os registros das posições onde os arquivos são encontrados na pilha representam a localidade temporal de acesso a estes arquivos. Quanto mais próximo do topo são registradas as posições, maior localidade de referência tem a carga. Ou seja, a ocorrência de pequenas distâncias entre as posições registradas e o topo da pilha indica que as requisições a um mesmo arquivo ocorrem com pequenos intervalos de tempo. Desta forma, pode-se considerar que a distribuição das distâncias de pilha observadas em um *log* de requisições Web serve para representar a localidade de referência destas requisições. Em [03, 23] é utilizada a distribuição *lognormal* para modelar a localidade temporal da carga Web.

5.3.5 Distribuição do Think Time

Em sistemas interativos como a Web, o intervalo de tempo entre duas requisições consecutivas de um mesmo usuário é chamado de *think time*. Este é o tempo necessário para que o usuário faça a leitura ou visualize a página que acabou de receber. Dependendo da definição, o *think time* pode ser acrescido do tempo necessário para

transferir o pedido HTTP através da Internet, do usuário até o servidor. Nos trabalhos [03, 23], através da análise de *logs* de requisições, este tempo foi caracterizado por uma distribuição de Pareto limitada com α=1,4 e k=1. O limite superior para esta distribuição é de 1800 segundos.

5.3.6 Número de Usuários Simultâneos

O número de usuários que têm acesso à Internet e, por sua vez, à Web é da ordem de dezenas de milhões. Desta forma, um servidor Web poderia ser acessado concorrentemente por milhões de usuários. Todavia, isto não ocorre divido à limitação imposta pela capacidade do enlace do servidor e, também, pelos diferentes interesses e inúmeras opções que o usuário da Web possui. Geradores de carga, como o SURGE, utilizam o conceito de usuários equivalentes para representar uma população de usuários acessando um servidor. No SURGE, um usuário equivalente (UE) é um processo que faz requisições ao servidor e aguarda pelas respostas. Entre cada requisição efetuada pelo UE existe um tempo de inatividade que corresponde ao *think time*. O tamanho das requisições e o *think time* exibem distribuição e relacionamento característicos de usuários reais da Web. Este modelo de geração de carga apresenta rajadas de requisições seguidas de longos tempos de inatividade (*think time*). Estas rajadas são conseqüência das páginas Web poderem ter diversas referências a outros objetos. Depois de um longo período de inatividade (*think time*) o usuário requisita uma página e o próprio *browser* se responsabiliza por requisitar, em um pequeno intervalo de tempo, os objetos referenciados. O intervalo de tempo entre a requisição de um objeto referenciado e outro é sempre inferior a um segundo [03].

A intensidade da carga gerada pelo SURGE é definida pelo número de UEs. Este número depende da capacidade da plataforma de software e hardware a ser avaliada e não tem relação direta com métricas do tipo requisições/segundo, bytes/segundo e taxa de utilização. O aumento do número de UEs pode não provocar um aumento proporcional nas métricas citadas acima. Este fato ocorre porque a geração da carga é feita como um sistema fechado onde os usuários somente fazem novas requisições após terem recebido a resposta da anterior. Se o número de UEs é muito alto para a plataforma avaliada, o tempo de resposta também torna-se elevado e o número de requisições atendidas por segundo tende a manter-se estável mesmo com novos aumentos do número de UEs. Ou seja, se o sistema está operando próximo à sua capacidade máxima, um aumento do número de UE não provocará um aumento

proporcional no número de requisições por segundo ou bytes por segundo. Contudo, é correto afirmar que quando o sistema está subutilizado, um aumento do número de UEs provocará um aumento proporcional nas métricas de desempenho.

O programa do SURGE é utilizado para gerar quatro arquivos no formato texto com as seguintes informações: o tamanho dos arquivos a serem requisitados, o número de arquivos por página, o *think time* e a seqüência na qual os arquivos devem ser requisitados. Esta seqüência representa também os aspectos de localidade temporal e popularidade de acesso dos arquivos. A seqüência de requisições gerada pelo SURGE não depende do número de UEs. O programa de simulação implementado neste trabalho faz a leitura dos arquivos gerados pelo SURGE e reproduz a carga de acordo com o número de UEs especificado.

5.4 Simulação da Internet

Fazer a modelagem de toda a topologia da Internet é um problema que ainda não foi solucionado [44]. As dificuldades para tal modelagem e simulação são apresentadas em [43], sendo elas: imenso tamanho; heterogeneidade técnica e administrativa; e elevada taxa de crescimento. Contudo, o estudo e modelagem de alguns aspectos da Internet são realizados por inúmeros trabalhos. O trabalho [42] faz a modelagem e simulação da Internet para avaliar a interação entre o servidor Web e a Internet. O trabalho [40] descreve uma metodologia para avaliar quais são os pontos críticos na interação entre o protocolo TCP e o protocolo HTTP. O trabalho [15] faz a modelagem do protocolo TCP para avaliar seu desempenho na transferência de pequenos arquivos pela Internet. Diversas configurações do TCP são testadas neste trabalho. O trabalho [39] descreve como medir o *bandwidth* e o congestionamento da Internet permitindo que seja feito o escalonamento distribuído de servidores Web. Semelhante pesquisa é realizada no trabalho [38], buscando-se medir o *bandwidth* de *links* intermediários na transferência de dados pela Internet. No trabalho [02] são realizados estudos para definir a relação entre o número de *hops* e o RTT, com o objetivo de utilizá-los para fazer escalonamento de servidores na Internet. A pesquisa realizada em [14] teve como meta produzir dados sobre a distribuição do RTT na Internet.

Embora nenhum dos trabalhos citados acima tenha realizado uma análise completa de toda a Internet, eles permitiram que certos aspectos da grande rede tenham sido decompostos e modelados. Semelhante à simulação da carga de serviço produzida

por geradores de carga como o SURGE, a simulação da Internet precisa ser restrita a uma população de um determinado número de usuários. Da mesma forma que é praticamente impossível simular a carga de serviço de todo o universo de usuários da Web, é também impossível modelar toda a Internet. No decorrer desta seção será apresentada uma modelagem para simular a Internet através de uma população de usuários que varia de 70 a 700.

A Internet, que serve como meio de transmissão para a Web, é composta por uma grande variedade de redes (hardware) e protocolos (software). Diversos são os fatores que contribuem para que haja na Internet uma enorme variabilidade da latência, dentre os quais podemos destacar:

- diferentes velocidades de conexão com os usuários [10];
- variações na velocidade dos *links* intermediários que conectam o usuário [38];
- quantidade de tráfego disputando o *link* [38, 39];
- número de *hops* entre usuário e servidor [02];
- carga e desempenho do provedor de acesso;
- carga e desempenho dos servidores Web;
- funcionamento dos protocolos TCP e HTTP [06, 07, 20, 40].

Baseando-se nestes fatores, a Internet foi simulada modelando-se o funcionamento dos protocolos TCP e HTTP/1.1 e fazendo-se uso dos seguintes parâmetros: RTT médio, *bandwidth*, quantidade de tráfego disputando os *links* e perda de pacotes devido a congestionamentos. A seguir serão apresentados valores obtidos em alguns trabalhos, os quais foram utilizados como base para a distribuição do RTT e do *bandwidth*. A partir de um tamanho de arquivo, de um RTT médio, de um *bandwidth* e da modelagem dos protocolos é calculado o tempo total de transmissão do arquivo.

O RTT é modelado a partir do RTT mínimo. Este, por sua vez, é o menor tempo possível, gasto por um único pacote ou datagrama, para ser enviado de um *host*, chegar à outro *host* e retornar. O RTT é determinado pela velocidade da luz, pelo retardo imposto pelos pontos de conexão (roteadores, protocolos, etc) e pelo atraso devido ao tráfego da rede. Desta forma, o RTT varia de acordo com a carga à qual a rede está sendo submetida. Quanto maior o tráfego na rede, maiores serão as filas nos roteadores e maior será a latência. Em geral, as medidas de RTT possuem desvio padrão alto. No trabalho [14] foi apresentado um estudo do RTT sobre 90 *links* que interligam *hosts* que foram agrupados em três classes, a saber: comerciais, acadêmicos e estrangeiros. Este

estudo foi realizado nos Estados Unidos. Os *links* para *hosts* comerciais e acadêmicos apresentaram os menores RTTs mínimos por estarem fisicamente mais próximos, a maioria dentro dos Estados Unidos e alguns no Canadá. Os *links* para *hosts* estrangeiros tiveram os maiores RTTs mínimos justamente por estarem mais afastados. Foram avaliados *hosts* de quatorze países. *Links* para *hosts* comerciais próximos tiveram um RTT mínimo que variou de 10 a 20 ms, os mais distantes tiveram um RTT mínimo entre 70 e 90 ms. Os *links* para *hosts* acadêmicos tiveram um RTT mínimo na faixa de 20 a 120 ms e os *links* para *hosts* estrangeiros entre 90 e 600 ms.

Com base nas informações do artigo [14] foram estabelecidas, para este trabalho, quatro classes de RTT mínimo, a saber: classe **X**, 20 ms; classe **Y**, 50 ms; classe **Z**, 90 ms e classe **W**, 280 ms. Foi considerado que 25% dos RTTs são da classe **X**, 35% da classe **Y**, 15% da classe **Z** e 25% da classe **W**. As classes **X** e **Y** têm por objetivo simular acessos de usuários próximos e medianamente próximos. As classes **Z** e **W** visam simular acessos de usuários geograficamente mais distantes.

Uma simulação mais realística do RTT envolve não apenas o RTT mínimo mas também as variações decorrentes do tráfego na rede, que pode apresentar situações de congestionamento. As condições de tráfego geram variações no RTT que podem ser simuladas por distribuição de cauda pesada como a de Pareto [14]. Para simular esta variabilidade, o RTT foi modelado por uma distribuição de Pareto limitada, que tem como parâmetro o RTT mínimo de cada classe. O RTT gerado foi limitado a oito vezes o RTT mínimo. Outro parâmetro exigido pela distribuição de Pareto, além do valor mínimo, é o grau de variabilidade, denominado alfa, que foi fixado em 1,4. Este modelo de simulação dos RTTs gerou medidas semelhantes às apresentadas no artigo [02], no qual, para 5262 *hosts* servidores, observou-se uma média de 241 ms e um desvio padrão de 435 ms para o RTT. A modelagem da Internet a partir destes valores gerou resultados coerentes também com o trabalho [46], onde foi demonstrado que o tempo médio de duração de requisições HTTP na Internet fica entre 2 e 4 segundos.

A velocidade de transmissão (*bandwidth*) e sua relação com o RTT são apresentadas na Tabela 4.

Velocidade de transmissão	Freqüência (%)	Classe de RTT
14 Kbps	2	90 e 280
28 Kbps	4	90 e 280
33 Kbps	12	90 e 280
56 Kbps	34	90 e 280
128 Kbps	19	20 e 50
1 Mbps	13	20 e 50
4 Mbps	9	20 e 50
10 Mbps	7	20 e 50

Tabela 4 : Velocidade de conexão do usuário e RTT associado.

Estas velocidades de transmissão foram obtidas de uma pesquisa realizada pelo *Graphics, Visualization, & Usability Center* do *Georgia Institute of Technology* com 2710 usuários de diversos países em outubro de 1998 [10]. É importante observar que a característica que motiva a proposta da política FCF é a variabilidade das condições de conectividade dos usuários. Acreditamos que dados mais recentes podem alterar a freqüência apresentada na Tabela 4 mas não acreditamos que isso implica numa diminuição da variabilidade das condições de conectividade. Pelo contrário, é provável que tenha havido um aumento da variabilidade, uma vez que enquanto já há redes de alta velocidade em operação, nada indica que houve progressos para os usuários com conexões lentas. Na simulação realizada, cada usuário é enquadrado em uma linha da Tabela 4, com as características definidas. As velocidades iguais ou acima de 128 Kbps são enquadradas somente nas classes X e Y de RTT. As velocidades abaixo de 128 Kbps somente nas classes Z e W.

5.5 Modelagem do TCP e HTTP/1.1

O TCP é o protocolo de transporte usado pela maioria das aplicações da Internet: SMTP, NNTP, *telnet*, FTP e HTTP. Ele proporciona um mecanismo confiável, ordenado e bidirecional de troca de bytes entre duas aplicações de quaisquer *hosts* da Internet. Além disso, o TCP proporciona controle de fluxo evitando que o receptor receba dados acima da sua capacidade, e também evita e controla congestionamentos na rede. O HTTP é um protocolo do nível de aplicação utilizado transferir dados através da Web.

Baseando-se no funcionamento do TCP e do HTTP, o tempo de transmissão de requisições HTTP pode ser decomposto em duas fases, a saber: fase de conexão e fase de transmissão [07, 11, 15]. O tempo da fase de conexão, que é determinado essencialmente pelo RTT, corresponde ao tempo entre o envio, pelo usuário, do

segmento de abertura de conexão até o início da transferência de dados da resposta HTTP pelo servidor. O tempo da fase de transmissão, que depende também do *bandwidth*, corresponde ao tempo decorrido desde o início da transmissão de dados pelo servidor até o recebimento do último byte pelo usuário.

O TCP estabelece conexões em um processo de três fases [07, 30]. Primeiro, o *host* interessado em iniciar a conexão (usuário) envia um segmento com valor um no bit SYN do campo *CODE BITS* do cabeçalho TCP. Então, o outro lado (servidor) responde enviando um segmento, aceitando o segmento que recebeu, e informando seu próprio número de seqüência. Finalmente, quem iniciou a conexão, envia um ACK confirmando o número de seqüência recebido. Logo após enviar um segmento de confirmação (ACK), a estação usuário envia o *GET da requisição HTTP*. Desta forma, o cálculo do tempo da fase de conexão é dado por: 1,5*RTT. Este cálculo é baseado nos trabalhos [06, 07, 15, 40]. A figura 7 mostra uma interação entre usuário e servidor utilizando o TCP.

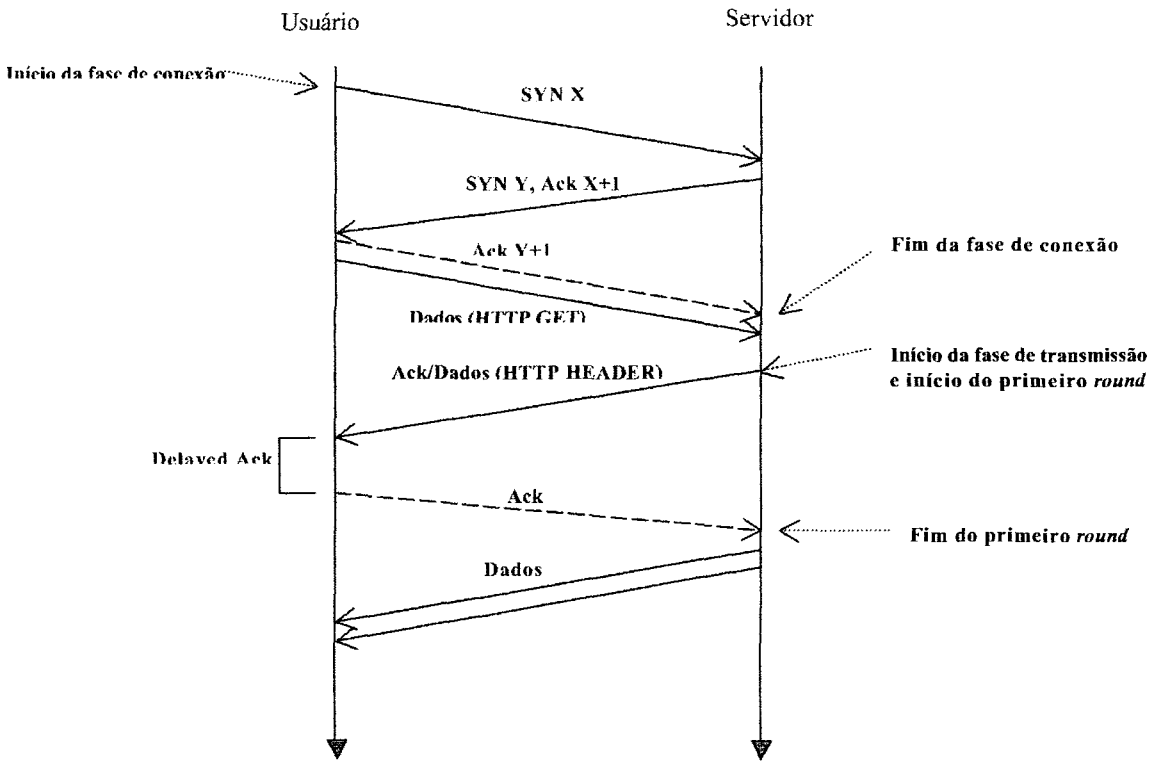

Figura 7 : Interação cliente/servidor pelo TCP.

Depois de decorrido o tempo de conexão, o servidor já deve ter recebido o GET da requisição HTTP e pode iniciar a transferência do arquivo solicitado. Neste ponto,

tem início a fase de transferência. O custo de transferência dependerá do *bandwidth* da conexão, da probalidade de perdas de segmentos e do número de *rounds* necessários para transmitir todos os dados.

Considerando os serviços de transferência confiável, controle de fluxo e congestionamento realizados pelo TCP, para fazer a modelagem do custo de transferência foi utilizado o conceito de *rounds*, conforme apresentado na figura 7. Para implementar os serviços citados, o TCP dá a cada byte do fluxo de dados um número de seqüência único. A seqüência de bytes a transmitir é dividida em segmentos com um tamanho máximo MSS (*Maximum Segment Size*). O MSS é tipicamente de 536 ou 1460 bytes [06, 15]. Cada segmento é colocado em um datagrama IP e enviado para o destino. Quando o receptor recebe um segmento, ele manda de volta um ACK que especifica o número do último byte recebido com sucesso. Contudo, o receptor nem sempre reconhece os segmentos imediatamente. As implementações do TCP variam muito neste sentido. A maioria utiliza um mecanismo de retardo permitindo que seja enviado apenas um ACK a cada dois segmentos recebidos com sucesso [15]. Este mecanismo é conhecido pelo nome de reconhecimentos atrasados (*delayed acks*). Caso não sejam recebidos dois segmentos completos o TCP aguarda um tempo que varia de 100 a 500 ms antes de enviar o ACK de confirmação. Desta forma, a estação usuário não envia um ACK logo após ter recebido o *Header* HTTP no primeiro segmento de dados vindo do servidor [15]. Como a conexão está no modo *slow start*, o servidor pode enviar apenas um segmento ao usuário.

Como mostrado na figura 7, logo após o recebimento do *Header* HTTP, o usuário aguardará um tempo de retardo conforme especificado no mecanismo de retardo. No momento em que o ACK confirmando o segmento que transmitiu o *Header* HTTP alcança o servidor termina o primeiro *round*. O número de segmentos que podem ser transmitidos em cada *round* é determinado pelos mecanismos de controle de fluxo e congestionamento do TCP. Estes controles são realizados através do uso de um esquema com duas janelas, a janela do receptor e a janela de congestionamento. A janela do receptor (*advertised window*), representada pela sigla *awnd*, depende do tamanho do *buffer* do receptor e é divulgada nos segmentos ACK. Para esta simulação, a janela do receptor está limitada por dois parâmetros. Primeiro, pelo tamanho do campo *Window* do cabeçalho do TCP. Este campo tem tamanho de 16 bits, o que corresponde a um tamanho máximo de janela de receptor de 65535 bytes. O segundo parâmetro é a velocidade de conexão do usuário. Por exemplo, se um usuário está

conectado através de um modem de 56 Kbps então a janela do receptor será fixada em 7 Kbytes. A velocidade de conexão do usuário é dada pela Tabela 4.

A segunda janela utilizada pelo TCP é denominada janela de congestionamento (*congestion window - cwnd*). A quantidade máxima de dados que o TCP pode transmitir em um dado momento é definida pela menor das duas janelas (*awnd* e *cwnd*). O TCP sempre envia o máximo de dados que as janelas permitem e espera pelo recebimento dos ACKs. Quando recebe um ACK, ele ajusta a *awnd* pela informação recebida no ACK. A *cwnd* é aumentada em um MSS por ACK recebido. Há dois modos nos quais é alterado o tamanho da *cwnd*: início lento (*slow start*) e prevenção de congestionamento (*congestion avoidance*) [15].

No *slow start*, a *cwnd* começa com um valor mínimo, normalmente um MSS, e aumenta em um MSS para cada ACK recebido. Com o mecanismo de retardo de ACK, para cada dois segmentos um ACK é recebido, assim uma *cwnd* de k segmentos resulta em k/2 ACKs. Desta forma, durante o *slow start* a *cwnd* cresce exponencialmente em um fator de 1,5 [15]. O *slow start* é utilizado no início da conexão, depois de um congestionamento e depois de períodos de ociosidade da conexão.

Quando a transferência inicia, a conexão está no modo *slow start*. Durante o primeiro *round*, depois de aberta a conexão, a transferência de dados está limitada pela janela de congestionamento (*cwnd*). A especificação do TCP descreve que a *cwnd* deve ter inicialmente o tamanho de um MSS, embora algumas implementações usam dois MSS e há propostas para usar uma *cwnd* inicial de três ou quatro MSS desde que não ultrapasse 4380 bytes [15]. Para esta simulação, o tamanho da *cwnd* no primeiro *round* foi de um MSS.

Quando a conexão entra no segundo *round*, o TCP age de acordo com a especificação do *slow start*, aumentando a *cwnd* em um MSS para cada ACK recebido. Este processo se repete até que todos os dados tenham sido enviados.

Considerando:

- que *cwnd*(i) seja a janela de um dado *round*(i);
- que a conexão esteja no modo *slow start*;
- que a janela do receptor (*awnd*) não foi alcançada;
- o mecanismo de retardo de ACK onde a cada dois segmentos recebidos um ACK é gerado;

para calcular o número de MSS da janela *cwnd*(i+1) no modo *slow start* foi utilizada a seguinte fórmula:

$cwnd(i+1) = cwnd(i) + cwnd(i) / 2$.

A *cwnd* é incrementada de acordo com o algoritmo do *slow start* até que ocorra uma perda de pacote por estouro de tempo. Quando uma perda for detectada, o TCP presume que a capacidade da rede foi ultrapassada e entra no modo de prevenção de congestionamento (*congestion avoidance*). Nesta fase da conexão, o tamanho da *cwnd* é definido pela regra *additive increase-multiplicative decrease*. Em situações de congestionamento (sinalizadas por perdas de pacotes) a *cwnd* é reduzida de forma exponencial. Do contrário ela é aumentada linearmente. Na ocorrência de congestionamento os pacotes que excederem as filas dos roteadores simplesmente são descartados. O TCP usa dois métodos para descobrir a perda de segmentos. O primeiro método é o *timeout* de retransmissão. Sempre que o TCP envia um segmento, um temporizador é iniciado para verificar quanto tempo leva para retornar o ACK do segmento. Para saber se ocorreu um estouro de tempo e a possível perda do segmento, o TCP faz estimativas do RTT a partir dos ACKs recebidos.

Em função da perda de pacotes, os algoritmos do *slow start* e *congestion avoidance* podem ser implementados em conjunto utilizando-se uma variável chamada *ssthresh* (*slow start threshold*). Nesta simulação, estes algoritmos foram implementados da seguinte forma: quando ocorrer a primeira perda de segmento por *timeout* da conexão, a variável *ssthresh* é inicializada com valor igual a *cwnd*/2 e a *cwnd* volta a ser de um MSS. Nos *rounds* seguintes a *cwnd* continua sendo aumentada pelo algoritmo do *slow start* (descrito acima) até alcançar o valor de *ssthresh*. Quando a *cwnd* atinge o valor de *ssthresh* a conexão entra no modo *congestion avoidance* e a *cwnd* passa a ser incrementada em apenas um MSS para cada *round* sem perda de segmentos. Toda a vez que ocorrer perdas de segmentos a *cwnd* será reduzida para *cwnd*/2. Sempre que o valor da nova *cwnd* for inferior a *ssthresh* seu tamanho é manipulado pelo algoritmo do *slow start*.

O segundo método utilizado pelo TCP para detectar perda de segmentos é denominado *Fast Retransmit* e não será modelado nesta simulação [30].

As perdas de segmentos, como vimos acima, afetam o tamanho da *cwnd* e fazem com que o TCP mude o mecanismo para controlar suas alterações. Além disso, considerando que o TCP provê um mecanismo seguro de transferência de dados, então ele deve retransmitir o segmento até que o receptor receba e devolva um ACK. Para esta simulação, a probabilidade de perda de segmentos foi de 5%. Esta probabilidade é condizente com os trabalhos [15, 34, 42]. No caso da ocorrência de perda de segmento,

é acrescido ao tempo de transmissão um valor igual ao *timeout* da conexão. O *timeout* inicia com um valor equivalente a duas vezes o RTT médio da conexão e é duplicado toda vez que uma perda de segmento é detectada. Esta abordagem é uma simplificação do algoritmo de *Karn* apresentado em [30]. No caso da perda de um segmento todos os dados do *round* são retransmitidos. Uma abordagem semelhante foi utilizada em [42]. Para finalizar, ao tempo de transferência é acrescido o tempo necessário para a efetiva transferência dos segmentos pela Internet. Este tempo foi calculado dividindo-se o tamanho do arquivo a transferir pelo *bandwidth* do usuário. Este *bandwidth* é apresentado na Tabela 4.

É importante observar que foi simulada a versão 1.1 do protocolo HTTP. Nesta versão, não há necessidade de abrir uma nova conexão para cada requisição. No caso do servidor Web Apache é possível configurar o número máximo de requisições transmitidas em uma única conexão e, também, o intervalo de tempo durante o qual a conexão deve permanecer aberta esperando por novas requisições. Esta configuração é realizada através das variáveis *MaxKeepAliveRequests* e *KeepAliveTimeOut*, respectivamente. Valores normalmente utilizados são 100 requisições para *MaxKeepAliveRequests* e 15 segundos para *KeepAliveTimeOut*. Nesta simulação também foram utilizados estes valores.

6. RESULTADOS

Neste capítulo são apresentados os parâmetros dos experimentos e os principais resultados obtidos. Estes resultados envolvem as características da carga de serviço, a avaliação da *starvation* de processos grandes e conexões lentas e a comparação entre três políticas de escalonamento distintas.

6.1 Descrição dos Experimentos e Parâmetros

A principal contribuição do trabalho é a proposta e simulação da política FCF e sua comparação com as políticas FIFO e SRPT. A política FIFO foi escolhida por ser padrão em servidores Web. SRPT foi escolhida porque o critério desta política também é adotado pela política FCF. SRPT tem um desempenho comprovadamente superior à FIFO. Nosso objetivo ao utilizá-la na comparação de resultados é verificar se há ganho de desempenho se forem aplicados critérios de escalonamento baseados também nas condições de conectividade.

Todos os experimentos foram realizados com três conjuntos de políticas para comparação. O primeiro conjunto, denominado FIFO nos gráficos deste trabalho, consiste na implementação da política FIFO para as filas de disco e de rede. A representação das filas do servidor pode ser visualizada na figura 5 da seção 5.2. O segundo conjunto, denominado FCF, implementa a política FCF nas filas de disco e de rede. O terceiro conjunto, chamado de SRPT, implementa a política SRPT nas filas de disco e rede. Para todos os conjuntos as requisições são admitidas na CPU pela política FIFO e processadas por PS.

A carga de serviço utilizada na simulação foi gerada tendo como base um servidor Web com 10.000 arquivos diferentes. O arquivo mais acessado teve 102.500 acessos. Em cada execução da simulação foram feitas 1.006.149 requisições ao servidor, sendo que deste total, 545.876 foram requisições diretas de usuários (arquivo base) e o restante foram de requisições a objetos referenciados para compor uma página (imagem, som, vídeo, entre outros). O tamanho médio dos arquivos foi de 14.072 bytes. Seguindo o modelo do SURGE discutido na seção 5.3.6, a intensidade da carga foi definida pelo número de usuários equivalentes (UEs). O número de UEs foi variado de 70 a 700. Sob o ponto de vista do servidor, cada uma das políticas foi avaliada com a

mesma carga de trabalho, isto é, os mesmos arquivos foram requisitados na mesma seqüência. Sob o ponto de vista do usuário, alguns usuários podem fazer mais ou menos requisições de acordo com a prioridade dada a eles pela política de escalonamento. Isto ocorre porque, no SURGE, o usuário que recebe resposta primeiro faz a próxima requisição da carga de serviço modelada para o servidor. Desta forma, se algum usuário tem maior prioridade, recebe respostas antes que os demais e, consequentemente, faz mais requisições.

A Internet apresenta as mesmas condições de conectividade para as três políticas. Isso quer dizer que, para os três conjuntos de testes, haverá o mesmo número de usuários enquadrados nas velocidades de transmissão e classes de RTT apresentados na Tabela 4. A probabilidade de perda de segmentos, congestionamento e modelagem do TCP e HTTP são idênticas para as três políticas. Contudo, cabe ressaltar que havendo usuários com prioridades diferentes poderá haver quantidades diferentes de requisições enquadradas nas classes definidas na Tabela 4. Como a política FCF atribui maior prioridade a conexões velozes, implicitamente ela está priorizando usuários deste tipo de conexão. Este fato será melhor comentado na seção 6.11.

A constante *beta* teve valor igual a dois na simulação. Desta forma, a partir da definição do menor arquivo da fila, um arquivo com até o dobro do tamanho poderá ter maior prioridade se for requisitado por um usuário com conexão mais rápida.

Para demonstrar as características da carga de serviço e permitir avaliar uma possível *starvation* sofrida pelas requisições a arquivos grandes foram utilizados três gráficos, a saber:

- percentual de requisições por tamanho de arquivo;
- percentual de bytes por tamanho de arquivo;
- percentual da demanda de serviço e percentual de requisições.

As métricas utilizadas para avaliar as políticas foram:

- tempo médio de resposta no servidor: tempo entre o momento da chegada da requisição no servidor e o término do seu processamento pelo servidor. Inclui os tempos que a requisição gasta em todos os recursos do sistema mais os tempos de fila nestes recursos.
- tempo médio de resposta no cliente: corresponde ao tempo decorrido desde o momento da submissão da requisição HTTP pelo cliente (usuário) até o recebimento de todos os bytes da resposta HTTP. Inclui o tempo gasto para a transferência da

requisição HTTP do cliente para o servidor, o tempo gasto no servidor e o tempo de transferência da resposta do servidor para o cliente pela Internet. Em parte, estes dois últimos tempos são contados em paralelo uma vez que, para uma mesma requisição, há processamento no disco do servidor e envio de dados pela Internet ao mesmo tempo. O tempo de transferência na Internet tem início quando os primeiros bytes da requisição são postos à disposição da Internet pela interface de rede. Não é considerado o tempo gasto pela máquina do cliente para exibir as informações no navegador (*browser*). A métrica é a média dos tempos de resposta no cliente de todas as requisições.

- tempo médio de resposta no cliente por tamanho de arquivo: é a média dos tempos de resposta no cliente para todas as requisições de uma classe de tamanhos. A divisão dos tamanhos é feita em múltiplos de dois, por exemplo: 2, 4, 8, 16 e assim por diante.
- tempo médio de resposta no cliente por velocidade de conexão: é a média dos tempos de resposta no cliente para todas as requisições de uma classe de velocidade de conexão. As classes de velocidades de conexão são exibidas na tabela 4 da seção 5.4.
- *slowdown*: é a soma dos tempos de fila no servidor para uma dada requisição dividida pela demanda de serviço desta requisição nos três recursos do sistema (CPU, disco e rede); *slowdown* médio é a média dos *slowdown* de todas as requisições.
- número médio de requisições pendentes: uma requisição é considerada pendente se ela não depende mais do processamento em nenhum dos recursos do servidor, mas ainda não foi totalmente transferida pela Internet. A requisição permanece como pendente enquanto estiver sendo transmitida pela Internet.
- tamanho médio das filas de disco, interface rede e CPU: indica o tamanho médio das filas do servidor.
- número médio de requisições atendidas por segundo: número total de requisições atendidas durante a simulação dividido pelo tempo de duração da simulação.
- taxa de serviço em bytes por segundo: número total de bytes servidos durante a simulação dividido pelo tempo de duração da simulação.
- tempo médio de Internet: tempo desde o momento em que a interface de rede torna disponível os primeiros bytes para transmissão na Internet até quando o usuário

recebe os últimos bytes do arquivo solicitado. Inclui o tempo de transmissão na Internet mais o tempo de permanência no servidor excluindo-se o tempo inicial de filas.

- tamanho médio da janela do TCP: corresponde à média dos tamanhos máximos da janela do TCP observada para cada requisição; a janela do TCP é limitada pela menor das janelas de congestionamento e do receptor.

Para as principais métricas também foram calculados percentis. Um percentil, também chamado de porcentil, é uma medida da posição relativa de uma unidade observacional em relação a todas as outras. O p-ésimo porcentil tem p% dos valores abaixo daquele ponto e (100 - p)% dos valores acima. No apêndice 1 é apresentada uma tabela com os percentis 80°, 90° e 99° divididos em função da carga (70 a 700 UE) e da política (FCF, FIFO e SRPT) para as métricas *slowdown*, tempo de resposta no servidor, tempo de resposta no cliente e tempo de Internet. O apêndice 2 mostra os percentis 80°, 90° e 99° para os tamanhos de arquivo, custo de CPU, custo de disco e custo de interface de rede, os quais não variam nem em função da carga, nem em função da política.

É importante ressaltar que muitos resultados das políticas FCF e SRPT são semelhantes pelo fato de serem políticas que compartilham o mesmo critério de escalonamento. Ambas visam priorizar as requisições menores. A diferença é que FCF considera também aspectos da Internet para realizar o escalonamento, conseguindo desta forma melhores resultados em métricas que dependem da Internet como, por exemplo, tempo de Internet, requisições pendentes e tempo de resposta no cliente. O objetivo da FCF é otimizar a interação entre servidor Web e Internet. A política SRPT, por ser a mais beneficiada pelas características da carga de serviço e por não considerar a Internet, tem melhores resultados em métricas relacionadas diretamente ao servidor como tempo de resposta no servidor e tamanho de filas. Contudo, deve ser observado que ambas as políticas (FCF e SRPT) apresentam resultados muito melhores que a política padrão de servidores Web (FIFO). O fato é que a política FCF, além de ter como objetivo melhores valores para métricas convencionais, visa também otimizar a interação entre servidor e Internet. Por exemplo, ela evita que o servidor empregue recursos desnecessários em conexões lentas, procurando manter o tempo de resposta no servidor compatível com o tratamento recebido na Internet.

O ajuste dos parâmetros tamanho e velocidade de conexão na política FCF é muito delicado. Na política FCF, o que indica maior ou menor peso das condições de conectividade é a constante *beta*. Quanto menor for esta constante mais próxima estará a

política FCF da política SRPT. Portanto, é possível aumentar a diferença entre as políticas aumentando o valor de *beta*. Mas até que ponto vale a pena aumentar o peso da velocidade de conexão, reduzindo o peso do tamanho da requisição? Deve haver um compromisso na variação dos parâmetros tamanho e velocidade da conexão. É importante utilizar um valor de *beta* que mantenha o ganho de desempenho obtido pela prioridade dada às requisições menores e, também, consiga vantagem com a prioridade dada às conexões mais rápidas. O fato é que com o valor utilizado para *beta* nesta simulação, a política FCF apresentou melhor resultado na principal métrica para avaliação de desempenho de um servidor Web – o tempo de resposta visto pelo cliente. Acreditamos que em um ambiente real, a deficiência nas métricas específicas para o servidor possa ser minimizada pelo efeito da redução do número de requisições pendentes, o que não pôde ser simulado neste trabalho.

6.2 Caracterização da Carga de Serviço

Para facilitar a avaliação de uma possível situação de *starvation* dos processos grandes, a carga de serviço foi classificada pelo tamanho dos arquivos em oito classes. O gráfico da figura 8 mostra o percentual de requisições em cada classe de arquivo. Podemos observar que a grande maioria das requisições é para pequenos arquivos. Mais de 90% das requisições são para arquivos menores que 32 Kbytes e, por outro lado, apenas 1,47% das requisições é para arquivos maiores que 128 Kbytes. Estes dados também podem ser confirmados no apêndice 2, onde são exibidos os percentis de tamanho de arquivo para 80°, 90° e 99°, os quais são respectivamente, 11.239, 31.191, 76.172 bytes, ou seja, 99% dos arquivos são menores que 76 Kbytes. Esta característica da carga Web será utilizada, no decorrer deste capítulo, para explicar a redução no tempo médio de resposta obtido pelas políticas FCF e SRPT.

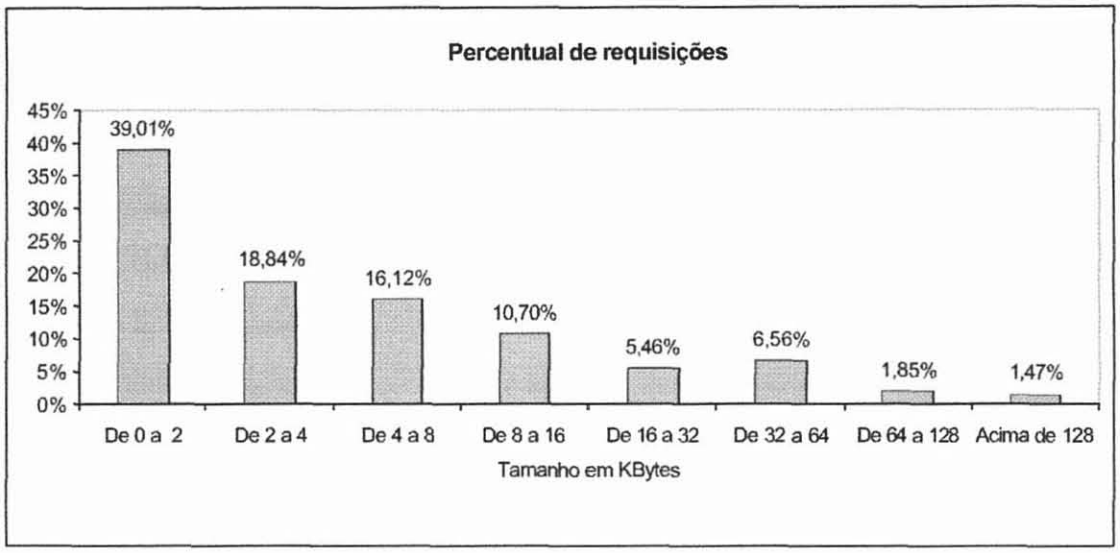

Figura 8 : Percentual de requisições por tamanho de arquivo.

No gráfico da figura 9 é apresentado o percentual de bytes em cada classe de arquivo. Inversamente proporcional ao percentual de requisições, podemos ver neste gráfico que as requisições para arquivos acima de 128 Kbytes representam quase 40% da carga imposta ao servidor, uma vez que a demanda de serviço é proporcional ao número de bytes. Esta propriedade da carga Web permite que seja utilizada uma política como SRPT sem que ocorra *starvation* dos processos grandes. Comprovações deste fato serão apresentadas no decorrer deste capítulo.

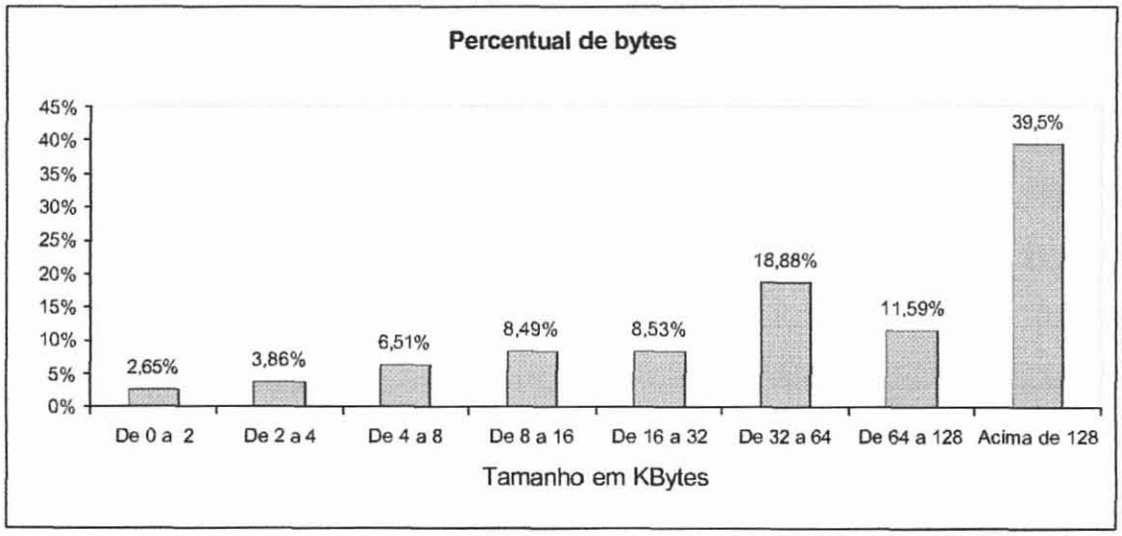

Figura 9 : Percentual de bytes por tamanho de arquivo.

Para concluir a caracterização da carga é apresentado no gráfico da figura 10 o percentual da demanda de serviço por tamanho de arquivo e por recurso do sistema. Verificamos novamente a inversão da demanda de serviço em relação ao percentual de requisições. Esta inversão é mais nítida na interface de rede onde a demanda de serviço é diretamente proporcional ao número de bytes. No caso das demandas de disco e CPU a inversão é menos evidente devido aos custos fixos, como por exemplo, tempo de *seek* e latência no disco, e tempo de *parser* e *log* na CPU.

No apêndice 2 são também exibidos os percentis para os custos de CPU, disco e interface de rede. Os percentis deste apêndice não apresentam variação em função da carga e da política. Isto ocorre porque todas as políticas foram avaliadas sob a mesma carga e portanto, sob os mesmos custos de CPU, disco e interface de rede.

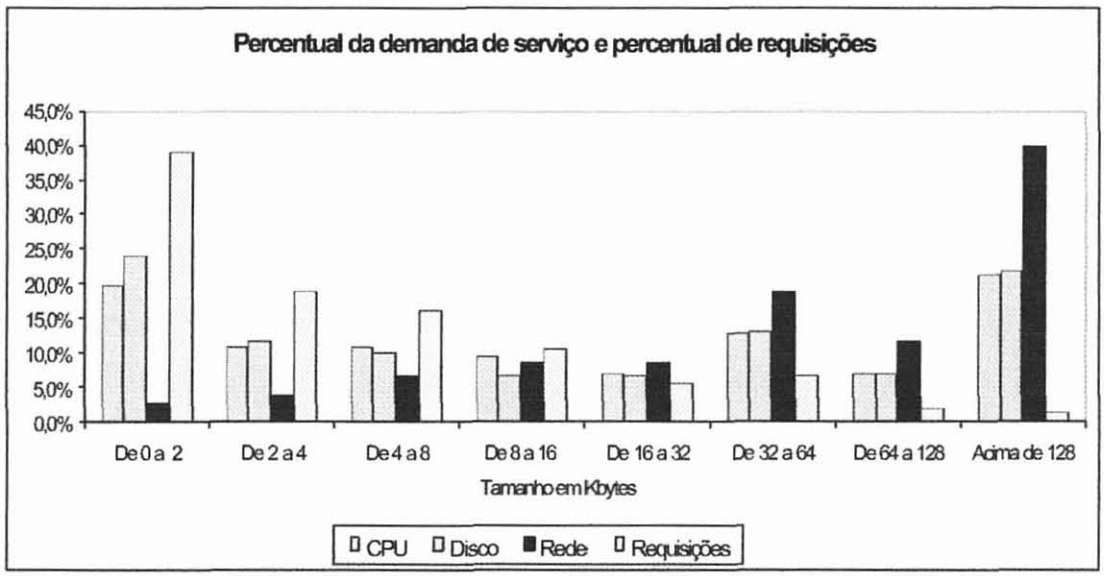

Figura 10 : Percentual da demanda de serviço e percentual de requisições.

6.3 Avaliação da *Starvation* Sofrida por Arquivos Grandes

Como podemos observar nos gráficos das figuras 8 e 9, a classe dos arquivos acima de 128 Kbytes representa apenas 1,47% das requisições, mas corresponde a 39,50% dos bytes a serem processados e transmitidos. Portanto, essa classe, apesar de ter um número reduzido de requisições, representa grande parte da carga imposta ao servidor e à Internet - considerando que a demanda de serviço é proporcional ao tamanho do arquivo [04, 17]. O fato de um pequeno número de requisições corresponder a grande parte da carga de serviço impede que processos grandes sejam demasiadamente prejudicados. Observamos nos gráficos das figuras 8 e 9, que cerca de 85% dos arquivos requisitados são menores que 16 Kbytes e representam apenas

21,50% da carga. Dessa forma, mesmo dando-se prioridade máxima a 85% das requisições (pequenos arquivos), o servidor ainda vai dispor de 78,50% dos recursos para tratar os 15% restantes das requisições. Considerando-se esta característica da carga Web, não há necessidade da utilização de prioridades dinâmicas, como a utilizada no artigo [09], para evitar que ocorra *starvation* dos processos grandes. Essa mesma linha de raciocínio é apresentada nos trabalhos [16, 17, 18, 19].

Para comprovar a não ocorrência de *starvation* apresentamos o gráfico da figura 11, onde podemos observar o tempo médio de resposta visto pelo cliente para 700 UEs. Este gráfico é apresentado em escala logarítmica e mostra que apenas as requisições para arquivos com mais de 128 Kbytes sofrem pequena penalização nas políticas FCF e SRPT em relação à FIFO. Deve-se observar que esta classe de tamanho de arquivo é responsável por apenas 1,47% das requisições. Além disso, a penalização destas requisições não é excessiva, o que pode ser melhor observado na Tabela 5.

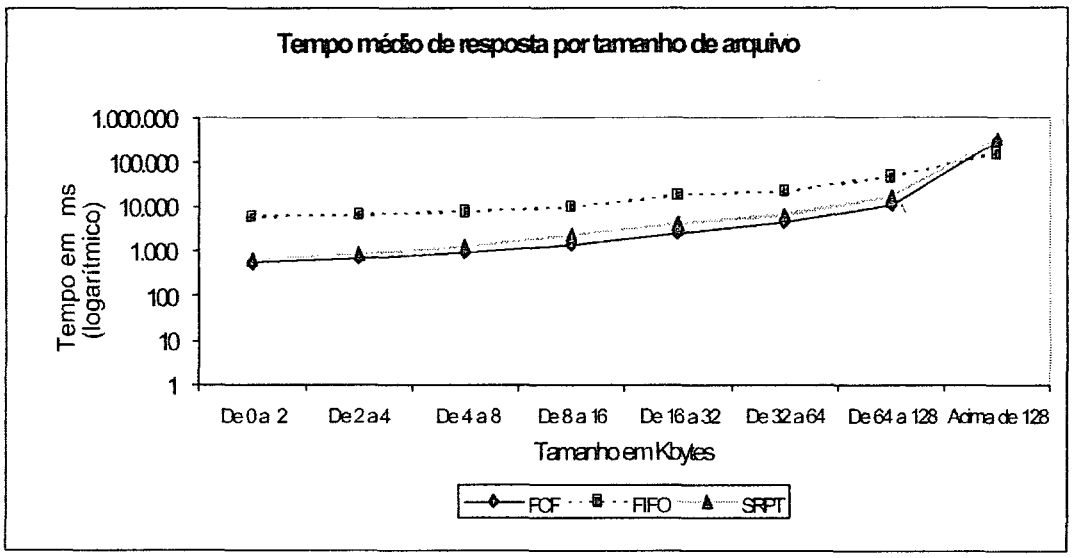

Figura 11 : Tempo médio de resposta por tamanho de arquivo para 700 UEs.

Tamanho em Kbytes	FIFO (ms)	FCF (ms)	SRPT (ms)
De 0 a 2	5.968	539	629
De 2 a 4	6.577	710	871
De 4 a 8	7.723	952	1.299
De 8 a 16	9.878	1.420	2.257
De 16 a 32	18.104	2.456	4.140
De 32 a 64	21.694	4.586	6.681
De 64 a 128	47.114	10.937	17.384
Acima de 128	157.103	287.488	319.044

Tabela 5 : Tempo médio de resposta visto pelo cliente para 700 UEs.

Na tabela 5 estão sendo exibidos os dados do gráfico da figura 11. Observamos que na política FCF as requisições para arquivos acima de 128 Kbytes possuem um tempo médio de resposta 82,99% superior ao da política FIFO. Isso demonstra que alguns processos estão sendo prejudicados em benefício de outros. Esse fato é perfeitamente aceitável em sistemas com recursos limitados, uma vez que não se pode dar benefícios a alguns processos sem que outros sejam penalizados. Deve ser evitado, contudo, que alguns processos sejam demasiadamente prejudicados a ponto de não serem concluídos ou levarem um tempo inaceitável para conclusão e, como vimos, isso não ocorre. Por outro lado, podemos observar grande redução no tempo médio das requisições para pequenos arquivos. Para arquivos com até 2 Kbytes, o tempo médio de resposta para FIFO é 1007,33% superior a FCF. Vale lembrar que quase 40% das requisições estão nesta classe de tamanho. Uma vez que a grande maioria das requisições são para pequenos arquivos, por exemplo, 84,67% são menores que 16 Kbytes, dar prioridade a estas requisições tem uma influência muito grande no tempo médio de resposta. Em servidores Web com políticas de escalonamento padrão (FIFO), as requisições para pequenos arquivos esperam pelo atendimento de requisições muito grandes, aumentando assim o tempo médio de resposta. Embora tenham sido apresentados apenas os tempos de resposta para a carga de 700 UEs, os demais casos apresentaram resultados muito semelhantes.

Observamos também, na Tabela 5, que a política SRPT apresenta tempo médio de resposta superior a FCF em todas as classes de arquivo. Em média, a política SRPT apresenta um tempo 39,87% superior ao da FCF. Este resultado será melhor comentado no decorrer deste capítulo.

6.4 Avaliação da *Starvation* Sofrida por Requisições de Conexões Lentas

O gráfico da figura 12 mostra o tempo médio de resposta visto pelo cliente em função da velocidade de conexão para 700 UEs. Observamos que os tempos acompanham as condições de conectividade, isto é, usuários de conexões mais rápidas têm melhores tempos de resposta para todas as políticas, mas muito mais marcante para FCF. Deve ser ressaltado, contudo, que em nenhum caso a política FCF apresenta tempo médio de resposta superior à FIFO. Isso deixa claro que não ocorre *starvation* das requisições de conexões lentas. Em relação à política SRPT, as conexões mais lentas apresentam um tempo de resposta superior, todavia, este é o propósito inicial da

política FCF, dar maior prioridade às conexões mais rápidas em detrimento às conexões mais lentas.

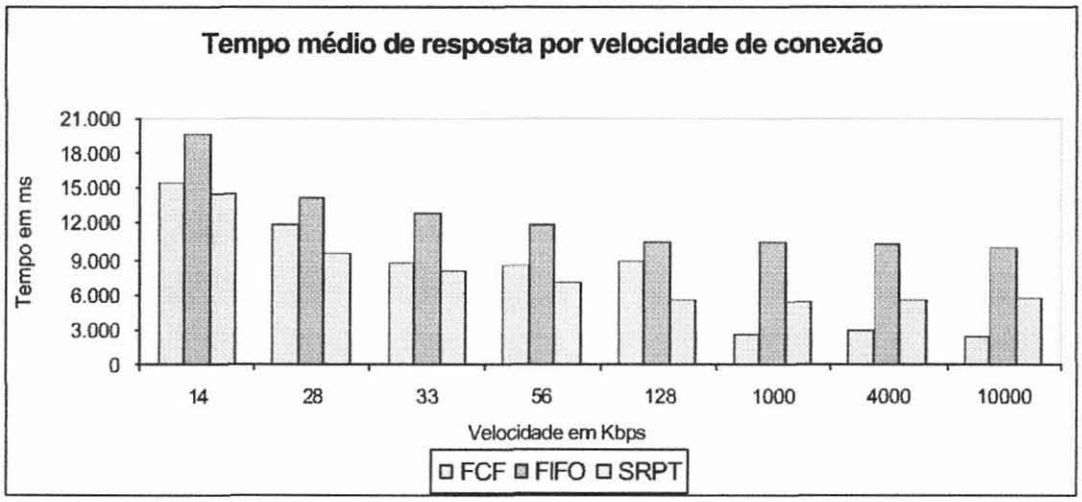

Figura 12 : Tempo médio de resposta por velocidade de conexão para 700 UEs.

Devemos frisar que todas as políticas apresentam um tempo de resposta proporcional à velocidade de conexão, porém em diferentes escalas. Isto ocorre porque o tempo de resposta visto pelo cliente inclui o tempo no servidor mais o tempo de transmissão na Internet. Este último tempo é proporcional à velocidade de conexão, fazendo com que todas as políticas apresentem tempo proporcional à velocidade de conexão.

6.5 *Slowdown*

Uma forma de avaliar a justiça de uma política de escalonamento é através da métrica chamada *slowdown*. O *slowdown* é calculado dividindo-se o tempo nas filas pela demanda de serviço nos três recursos do sistema (CPU, disco e rede). Um *slowdown* baixo significa que o servidor está tratando as requisições de forma compatível com seu tamanho.

O *slowdown* avalia a justiça das políticas em relação ao tempo de espera no servidor. O objetivo é manter este valor o mais baixo possível, garantindo boa relação entre o custo da requisição e a espera nas filas. O gráfico da figura 13 apresenta os valores desta métrica.

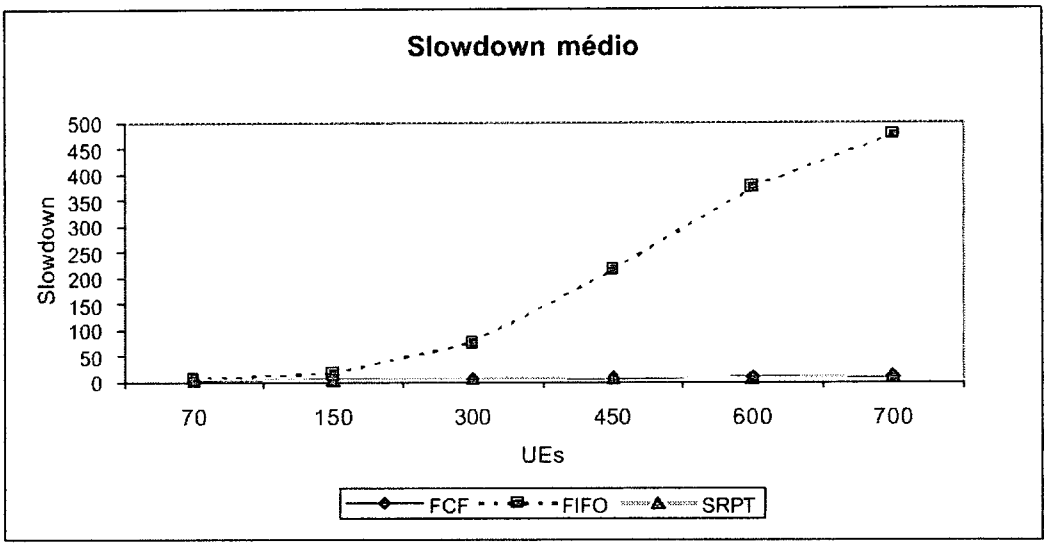

Figura 13: *Slowdown* médio.

Podemos observar que as políticas FCF e SRPT proporcionam maior justiça no tratamento das requisições. No pior caso, com 700 UEs, o *slowdown* da FIFO é cerca de quarenta vezes maior que o da FCF. A obtenção de um *slowdown* médio bastante baixo é decorrência, principalmente, do maior peso que a política de escalonamento atribui ao tamanho da requisição. Dessa forma, evita-se que pequenas requisições permaneçam durante muito tempo no servidor. No caso da FIFO, o atendimento de uma requisição muito grande obriga às requisições, comparativamente muito pequenas (que são a maioria), a aguardar na fila um tempo dezenas ou centenas de vezes maior que seu tamanho, o que contribui para aumentar o *slowdown*. Deve-se ressaltar que a política FCF teve um *slowdown* 1,42% superior à SRPT. Este comportamento é perfeitamente normal, uma vez que a política FCF não define a prioridade apenas com base no tamanho do processo, mas também, com base na taxa de transmissão. Desta forma, requisições com processos pequenos podem ter que esperar mais tempo porque são de usuários com conexões lentas. Contudo, esta diferença é muito pequena se comparada com a diferença que há para a política FIFO. O *slowdown* de FIFO é cerca de 2000% em média superior ao da política FCF. Este percentual também é observado no apêndice 1 para o percentil 99º e carga de 700 UEs. Neste caso, o percentil 99º do *slowdown* para a política FCF é 105,66 e para a política FIFO é 2.141,78. Uma diferença de mais de 2000% para 99% das requisições.

6.6 Requisições Pendentes no Servidor

Uma vantagem observada na política FCF, decorrente da prioridade dada às requisições de conexões velozes, refere-se ao número médio de requisições pendentes no servidor. Requisições pendentes são aquelas cujas conexões já poderiam ter sido fechadas pelo servidor, mas que ainda não foram concluídas devido à demora na transferência pela Internet. Para requisições de conexões lentas, mesmo que o servidor disponha de recursos para concluir seu processamento, isso não ocorre pois o envio de segmentos e recebimento de ACKs se prolonga por um tempo maior, obrigando o servidor a manter a conexão aberta. Por outro lado, priorizando-se conexões mais velozes, permite-se que as requisições que são transmitidas de forma mais rápida também sejam atendidas mais rapidamente pelo servidor. Em conseqüência disso, temos um menor número de requisições pendentes no servidor, como mostra o gráfico da figura 14. A política FIFO apresenta um número médio de requisições pendentes cerca de 30% superior à política FCF. Os motivos da redução acentuada observada entre 300 e 450 UEs serão comentados na seção 6.11. Neste gráfico também podemos constatar que a política SRPT, por não considerar os aspectos da Internet, mantém um número de requisições pendentes muito próximo ao da política FIFO.

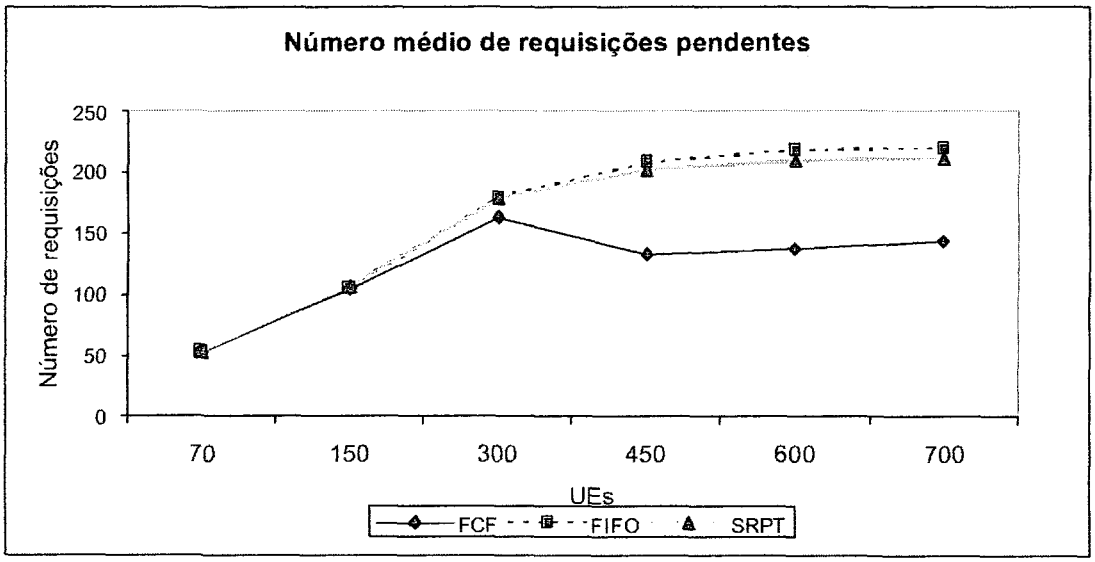

Figura 14 : Número médio de requisições pendentes.

A redução do número de requisições pendentes contribui para reduzir o número de trocas de contexto, o tamanho das estruturas de controle e a quantidade de memória necessária. Porém, estas métricas não puderam ser avaliadas no ambiente de simulação

utilizado. A avaliação destas métricas somente seria possível mediante a implementação real da política de escalonamento.

A comprovação de que o excesso de requisições pendentes prejudica o desempenho de servidores Web foi apresentado no trabalho [34]. Segundo os autores daquele trabalho, a redução no desempenho foi provocada justamente pelo aumento das trocas de contexto e pela necessidade de mais memória para gerenciar um maior número de requisições pendentes. O estudo apresentado no trabalho [05] também comprovou este fato. Nele os autores afirmaram que o problema das requisições que apresentam grandes latências de rede é que o servidor precisa manter um processo ocupado com a requisição até que o último segmento de dados seja enviado ao usuário. Isto reduz o desempenho do servidor. Fato semelhante foi apresentado no trabalho [32].

6.7 Tempo Médio de Resposta no Servidor e no Cliente

As métricas mais importantes para a avaliação da nova política de escalonamento são o tempo médio de resposta no servidor e o tempo médio de resposta visto pelo cliente. O gráfico da figura 15 mostra o tempo médio de resposta no servidor. Os valores apresentados correspondem à demanda de serviço nos três recursos do sistema (CPU, disco e rede) mais o tempo gasto nas filas dos respectivos recursos. Como podemos observar, os valores da opção FIFO são superiores ao da FCF, que por sua vez, são ligeiramente superiores ao da SRPT. Em média, esta métrica para a opção FIFO é 76% superior à FCF, e 95% superior à SRPT. No apêndice 1 nós podemos observar também que quando o servidor está operando com 80% de sua capacidade (300 UEs) a política FIFO apresenta o 80º percentil cerca de 17 vezes superior às demais políticas. Quando a utilização do servidor se aproxima da utilização máxima (450 UEs) o 80º percentil de FIFO chega a ser 44 vezes superior às demais políticas. Mesmo beneficiando enormemente 80% das requisições, pode-se observar que para a maior carga simulada no servidor (700 UEs) o 99º percentil de FIFO ainda é 18 vezes superior às demais políticas. O que garante a redução no tempo médio de resposta no servidor é o atendimento prioritário das requisições menores. Como este tipo de requisição corresponde à maioria das requisições e representa pequena fração da carga de serviço, reduzir seu tempo de resposta tem uma influência muito grande no tempo médio de resposta no servidor.

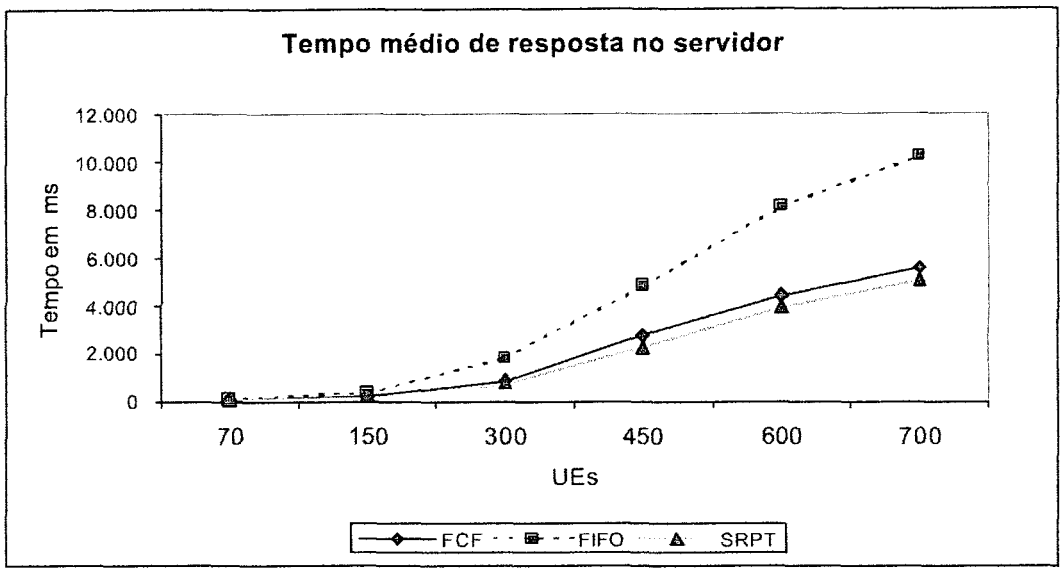

Figura 15 : Tempo médio de resposta no servidor.

Com relação à política FCF, uma vez que a prioridade é calculada também em função da taxa de transmissão, o que observamos é uma pequena queda de desempenho se comparado à SRPT. Contudo, esta queda de desempenho não é repassada ao tempo médio de resposta visto pelo cliente. Este fato pode ser constatado no gráfico da figura 16. Este gráfico apresenta o tempo médio de resposta observado pelo cliente. Os valores apresentados incluem o tempo de resposta no servidor mais o tempo de transferência na Internet. Não é computado o tempo gasto pela máquina do usuário para exibir os dados transmitidos.

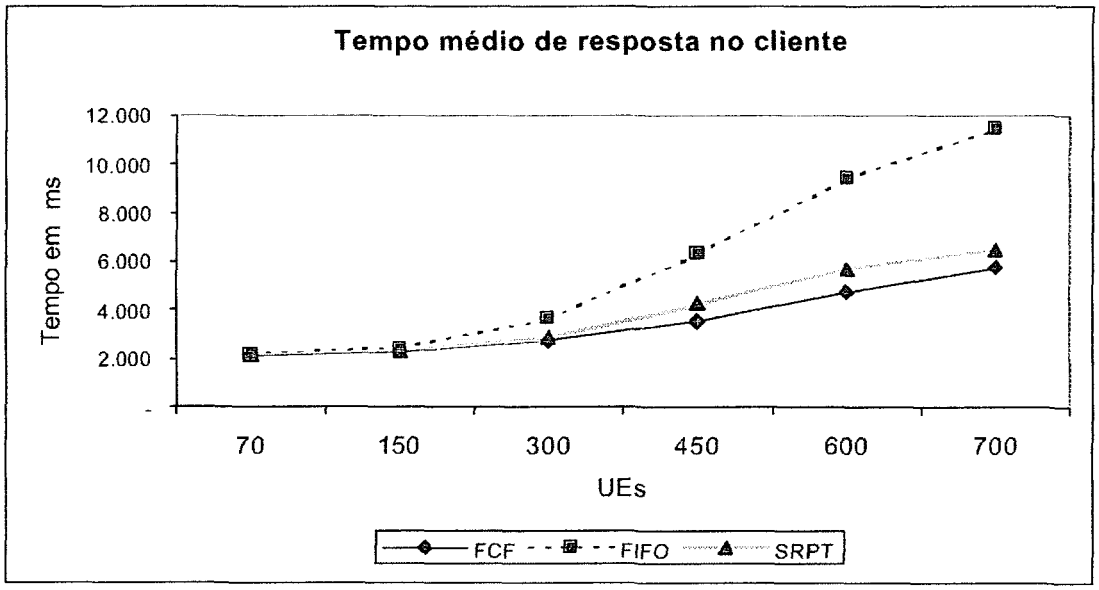

Figura 16 : Tempo médio de resposta para o cliente.

Neste gráfico observamos que a política FIFO novamente mostra o pior desempenho e que SRPT apresenta um desempenho intermediário. Por outro lado, o melhor desempenho é apresentado pela política FCF. As políticas FIFO e SRPT apresentam um tempo médio de resposta visto pelo cliente, respectivamente, 53% e 10% superior à FCF. Dois fatores contribuem para isso. Primeiro, a política FCF ajusta o tempo de resposta no servidor de acordo com o tempo de transmissão na Internet, ou seja, requisições que podem ser transmitidas com maior rapidez recebem mais recursos no servidor. Isso ajuda a explicar porque o tempo médio de resposta no servidor é menor para a política SRPT e o mesmo não ocorre com o tempo médio de resposta visto pelo cliente. O fato é que não adianta tratar com maior rapidez no servidor requisições que não terão um tratamento semelhante na Internet. Priorizar requisições de conexões lentas significa aumentar o número de requisições pendentes, como observado no gráfico da figura 14. Dar prioridade às requisições de conexões mais velozes contribui diretamente para melhorar o tempo de resposta visto pelo cliente. Este fato também pode ser observado no apêndice 1, onde, para a carga de 700 UEs, o percentil 99º do tempo de resposta no cliente em FIFO é mais de três vezes superior ao observado em FCF. Além disso, o tempo observado na política SRPT para a mesma carga e percentil gira em torno de 40% superior ao da política FCF.

O segundo fator que explica o melhor tempo médio de resposta visto pelo cliente na política FCF é o aumento do número de requisições de usuários com conexões velozes. Uma vez que ocorre a redução do tempo médio de resposta para este tipo de usuário, a tendência é que eles façam um maior número de requisições comparativamente aos usuários de conexões lentas. Embora o aumento do número de requisições de usuários com conexões velozes não seja expressivo, sem dúvida ele contribui para a redução no tempo médio de resposta visto pelo cliente.

6.8 Taxa de Serviço

O gráfico da figura 17 apresenta o número médio de requisições por segundo atendidas pelo servidor. Nele observamos uma pequena melhora de FCF e SRPT em relação à FIFO. FCF e SRPT têm praticamente os mesmos resultados com uma pequena vantagem de FCF. Acreditamos que em um ambiente real esta diferença possa ser maior devido a fatores que não puderam ser simulados como, por exemplo, o efeito do *cache* de disco, o efeito das trocas de contexto e o efeito da quantidade de memória utilizada.

Considerando que a política FCF mantém um menor número de requisições pendentes, estes fatores influenciam positivamente esta política. Contudo, eles não foram simulados devido à complexidade e falta de dados experimentais que permitissem o uso de distribuições na sua modelagem.

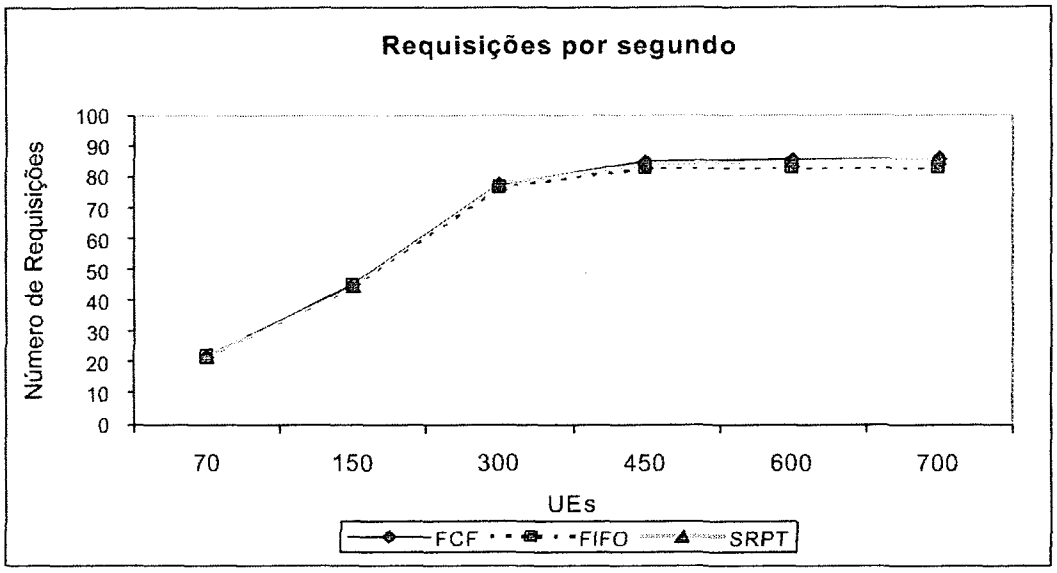

Figura 17 : Número de requisições por segundo.

Outro fator que contribui para que a diferença observada neste gráfico não seja maior é que os resultados apresentados são de uma simulação com término baseado em um número de requisições predeterminado (1.006.149), e não com término baseado em um tempo predeterminado. Na simulação de um número fixo de requisições, foi observado que embora a política FCF atenda mais rapidamente à maioria das requisições, o processo de simulação permanece em execução por um tempo maior, devido a um pequeno número de grandes requisições destinadas a usuários com conexões muito lentas, que ficam para o final. Por exemplo, uma requisição de um arquivo com 10 MBytes para um usuário com conexão de 14,4 Kbytes. Até que a transmissão desta requisição seja concluída, o processo de simulação não pode ser concluído e, durante este tempo, os recursos do sistema são subutilizados. A probabilidade de ocorrência de grandes requisições com conexões lentas no final da simulação é muito maior para a política FCF, uma vez que esta política prioriza justamente requisições pequenas feitas através de conexões velozes. Simulações com um intervalo de tempo predeterminado foram realizadas e apresentaram resultados melhores para a métrica número de requisições por segundo. Contudo, não achamos

justo apresentar os resultados deste tipo de simulação porque, desta forma, no final de um intervalo de tempo predeterminado restariam nas filas apenas as grandes requisições, o que afetaria erroneamente todas as métricas apresentadas neste trabalho.

Um detalhe importante do gráfico da Figura 17 é a carga equivalente à taxa de serviço máxima do sistema. Este ponto está situado entre 300 e 450 UEs. A partir deste ponto o sistema praticamente não mostra alterações no número de requisições por segundo. Este fato, como analisado na seção 5.3.6, ocorre porque a geração da carga é feita como um sistema fechado onde os usuários somente fazem novas requisições após terem recebido a resposta da anterior. Se o número de UEs é muito alto, o tempo de resposta tende a elevar-se e o número de requisições atendidas por segundo tende a manter-se estável mesmo com novos aumentos do número de usuários.

O gráfico da figura 18 mostra a taxa de serviço em bytes por segundo servidos pelo sistema. Observamos também uma pequena melhora para as políticas FCF e SRPT. Um fator que contribui para que a diferença entre SRPT e FIFO não seja maior é que o *throughput* de um servidor Web não depende exclusivamente do tempo de resposta no servidor. Ele depende também do tempo de transferência da Internet e da taxa de chegadas de requisições, esta última é definida pelo *think time* do usuário. A partir dos resultados de todas as simulações foi calculado o *think time* médio, cujo valor foi 3,32 segundos. O tempo médio de Internet de 2,95 segundos. O tempo médio de resposta no servidor foi de 2,87 segundos. Considerando que o usuário somente submete uma nova requisição após ter recebido a resposta da última, e depois de decorrido o *think time*, vemos que o tempo de resposta no servidor contribui em apenas 31,41% do tempo entre requisições (3,32+2,95+2,87=9,14 para 2,87 = 31,41%). Desta forma, a redução no tempo médio de resposta no servidor não permite visualizar um aumento muito grande no *throughput* do sistema como um todo.

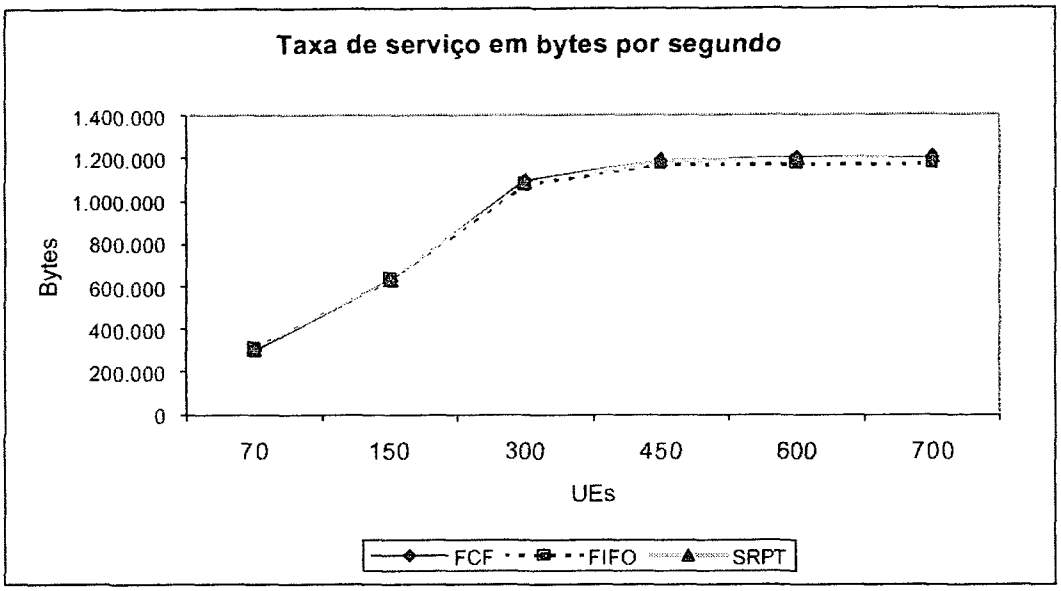

Figura 18 : Número médio de bytes por segundo.

6.9 Tamanho das Filas

O gráfico da figura 19 mostra o tamanho médio da fila da interface de rede. Observamos que FIFO mantém a fila muito maior do que SRPT, que, por sua vez, apresenta uma tendência de crescimento bem inferior. Isto ocorre porque a demanda de rede é diretamente proporcional ao tamanho dos arquivos, que seguem distribuição de cauda pesada. Neste tipo de distribuição a grande maioria das requisições é para pequenos arquivos e a pequena fração das requisições de arquivos grandes representa a maior parcela da carga imposta ao servidor. Como a política SRPT prioriza requisições menores, ficará na fila apenas a pequena fração de requisições para grandes arquivos, o que produz, consequentemente, filas menores. No caso da FIFO, a fila será maior pois as requisições pequenas ficarão esperando por serviço atrás das requisições grandes.

A política FCF apresentou um tamanho médio da fila de interface de rede 16% superior à SRPT. Isso demonstra que, quando a carga segue distribuição de causa pesada, o critério da quantidade de serviço na política de escalonamento é essencial para a métrica do tamanho da fila.

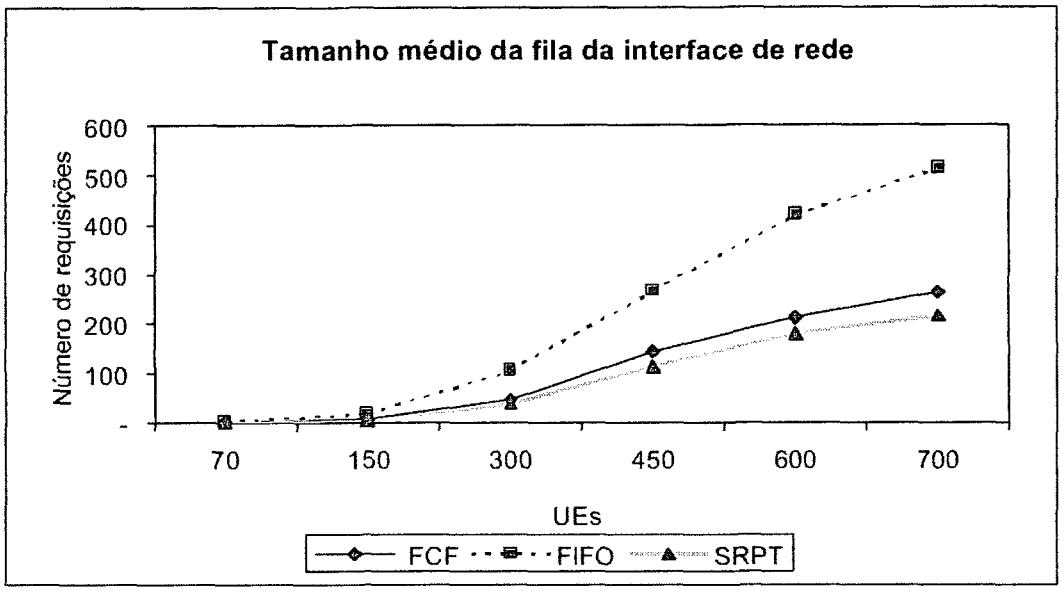

Figura 19 : Tamanho médio da fila da interface de rede.

O gráfico da figura 20 mostra o tamanho médio da fila de disco. Observamos que a diferença entre as políticas é ligeiramente menor do que a observada na fila da interface de rede. A diferença média entre as políticas SRPT e FIFO na fila de disco é de 45% e na fila da interface de rede é de 151%. Isto ocorre porque a demanda de disco não apresenta exatamente a mesma distribuição dos tamanhos de arquivo. No disco, a demanda é afetada pelos custos fixos como o tempo de *seek* e latência. As requisições menores passam a ter um custo maior e mais homogêneo. Este fato também pode ser observado no gráfico da figura 10. Isso reduz a variabilidade da carga de serviço aumentando o tamanho da fila. Acreditamos que em um ambiente real, o efeito do *cache* de disco reduza os custos fixos, o que torna a demanda de disco diretamente proporcional ao tamanho dos arquivos. Conseqüentemente, haverá redução no tamanho da fila de disco para as políticas SRPT e FCF. Um detalhe importante a ser observado é que o custo médio de disco é inferior ao custo médio da interface de rede. Isso explica porque o tamanho médio da fila de disco, considerando todas as políticas, é de 90 e da interface de rede é de 142.

Em máquinas utilizadas hoje como servidores Web, todos os arquivos usados nesta simulação poderiam ficar no *cache* de disco. Considerando 10.000 arquivos diferentes com tamanho médio de 14 Kbytes, seriam necessários apenas 136 Mbytes de memória RAM para que todos os arquivos fossem armazenados no *cache*. O padrão

atual de memória RAM em máquinas de servidores Web está entre 256 Mbytes e 4 Gbytes [47].

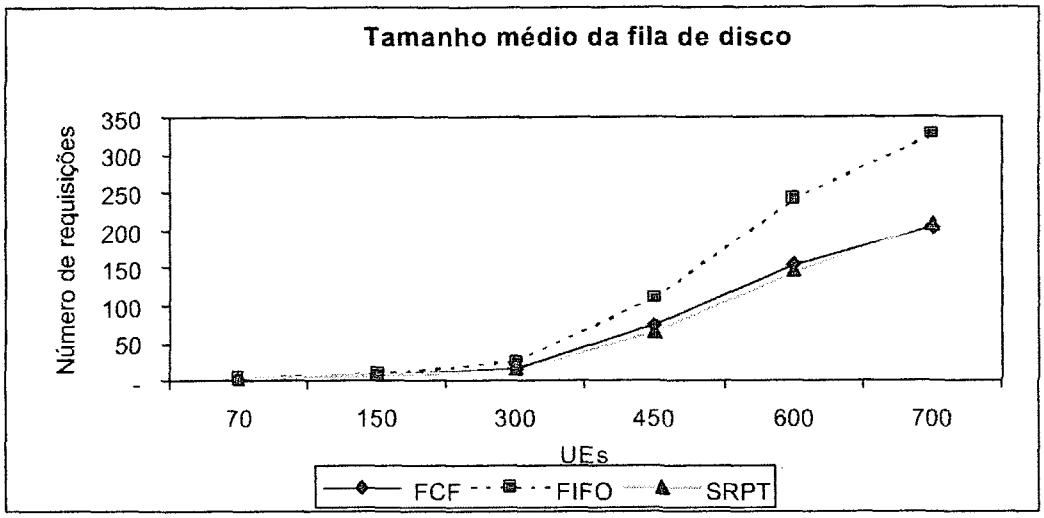

Figura 20 : Tamanho médio da fila de disco.

O gráfico da figura 21 mostra o tamanho médio da fila de CPU. Observamos que, quando o sistema atinge a utilização máxima, o tamanho da fila de CPU para as políticas FCF e SRPT tende a se estabilizar. O ponto de utilização máxima do sistema encontra-se entre 300 e 450 UEs. A mesma situação deveria ser observada para as três políticas uma vez que, em todas, as requisições são admitidas na fila de CPU de acordo com FIFO e processadas por PS. Porém, no caso da opção FIFO, o tamanho da fila de CPU apresentou um comportamento diferente, como pode ser observado no gráfico da figura 21. A explicação para este fato é a forma na qual as requisições são geradas. Quando a simulação inicia é gerado um número de requisições igual ao número de UEs especificado. Embora tenha sido utilizado um espaçamento médio de 25 milissegundos entre estas requisições iniciais, elas representarão, para o servidor, uma grande rajada de requisições. A partir destas requisições iniciais, as novas requisições são realizadas de acordo com a resposta do sistema (tempo de resposta no servidor, tempo de transferência na Internet e *think time* do usuário). Contudo, uma nova rajada de requisições, talvez um pouco menos intensa, vai ocorrer após o tempo médio de resposta do sistema. No entanto, as políticas SRPT e FCF reduzem o efeito deste ciclo de rajadas de requisições quando priorizam requisições menores, reduzindo drasticamente seu tempo de resposta. Isto pode ser comprovado na Tabela 5, onde podemos ver que o tempo médio de resposta em FIFO é muito mais homogêneo que o

apresentado em SRPT e FCF. No caso da política FIFO, como os tempos de resposta são mais homogêneos, a tendência é que o ciclo de rajadas aconteça com maior intensidade provocando, consequentemente, maior fila de CPU. É importante ressaltar que o tamanho da fila de CPU não afeta o tamanho das demais filas porque as requisições são passadas para o disco somente após terem sido totalmente processadas pela CPU. No modelo adotado, a CPU não impõe contenção às requisições que estão sendo processadas no disco ou na interface de rede.

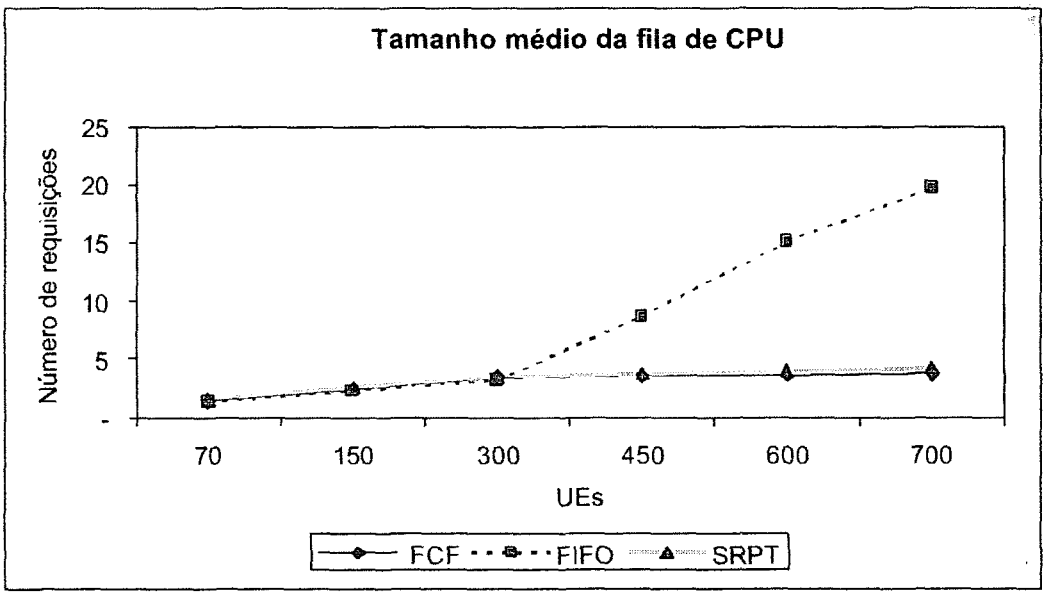

Figura 21 : Tamanho médio da fila de CPU.

6.10 Tempo de Internet

Outra métrica calculada para comparar as políticas foi o tempo gasto para transferir os arquivos pela Internet. O objetivo desta métrica, chamada de tempo de Internet, é avaliar o atraso imposto pelo servidor na transferência do arquivo pela Internet. Ela considera aspectos inerentes à Internet (protocolo TCP, protocolo HTTP, RTT, tráfego, perda de pacotes e *bandwidth*) mais o atraso imposto pelo servidor Web. Além do tempo inerente à Internet, esta métrica foi calculada levando-se em consideração a disponibilidade de recursos no momento em que ACKs de confirmação são recebidos, ou seja, ela considera o tempo de resposta no servidor excluindo o tempo inicial de filas. O tempo de Internet começa a ser contado quando os primeiros bytes do arquivo a ser transferido são postos à disposição da Internet pela interface da rede, ou seja, começa quando a Internet inicia a transferência da resposta HTTP. Ele termina

quando o usuário recebe completamente o arquivo solicitado. Deve-se ressaltar que todas as políticas foram avaliadas sob as mesmas condições de conectividade.

O gráfico da figura 22 mostra o tempo médio de Internet. Podemos observar que, para a política FIFO, este tempo é em média 25% superior à FCF. Isto ocorre pelo fato de que mesmo a Internet podendo transmitir o arquivo em tempo inferior, ela não o faz porque os dados deste arquivo ainda não foram disponibilizados pela interface de rede. Na prática, o processamento da requisição está esperando em alguma fila do sistema, seja na fila de processamento das primitivas *write* (CPU/Disco), na fila de mensagens de controle da saída do TCP, na fila de datagramas IP, ou mesmo na fila de pacotes da interface de rede [27]. Este fato justifica a importância de dar prioridade a requisições de conexões mais velozes. As requisições destas conexões devem ser atendidas prioritariamente para manter um tratamento compatível com aquele recebido na Internet. É neste ponto que a política FCF proporciona um melhor aproveitamento da Internet. Além disso, diminuir o tempo de duração da conexão contribui para a escalabilidade da rede, pois libera recursos para outras requisições.

No caso da política SRPT, observamos uma perda média de desempenho que gira em torno de 9% em relação à FCF. Isto acontece porque SRPT trata indistintamente requisições pequenas sem considerar sua velocidade de conexão. Desta forma, requisições pequenas de conexões lentas recebem maior prioridade mas não são transferidas rapidamente pela Internet. Por outro lado, requisições um pouco maiores, mas de conexões velozes, recebem menor prioridade fazendo com que seu tempo de resposta seja maior devido ao atraso ocorrido no servidor. Em outras palavras, a política SRPT não garante um tratamento da requisição compatível com sua velocidade de conexão.

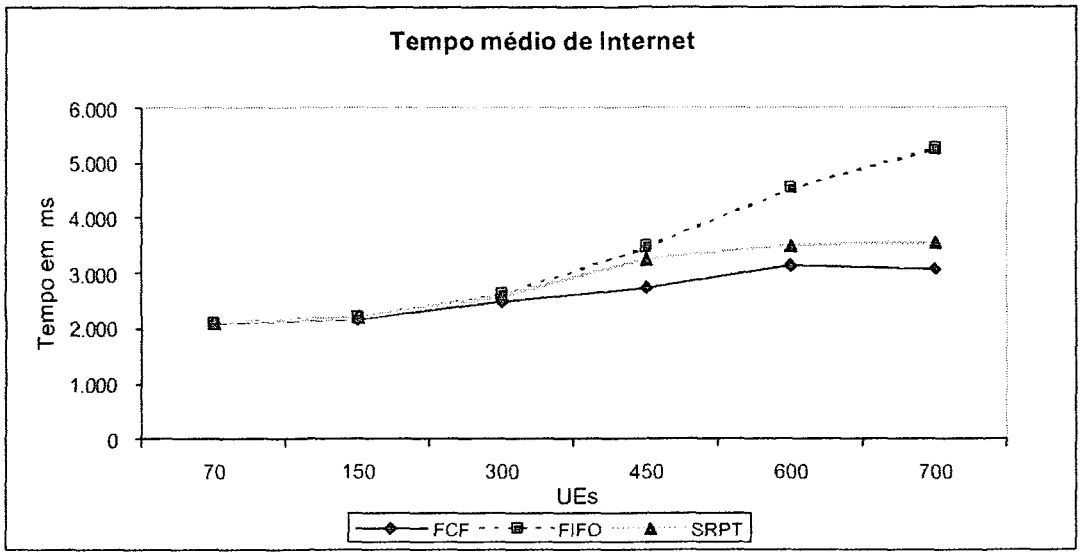

Figura 22 : Tempo médio de Internet

6.11 Janela de Congestionamento

A janela de congestionamento (*cwnd)* e a janela do receptor (*awnd*) definem a quantidade de bytes que o TCP pode transmitir em um dado momento. A *awnd* foi definida em função das condições de conectividade dos usuários (apresentada na Tabela 4) não tendo sido alterada no decorrer da simulação. A *cwnd*, por sua vez, tem seu tamanho alterado de acordo com os algoritmos *slow start* e *congestion avoidance.* O tamanho médio desta janela para a política FCF pode ser observado no gráfico da figura 23. O tamanho médio da *cwnd* na política FCF elevou-se na medida que a carga aumentou. Situação contrária ocorreu nas outras duas políticas, as quais mantiveram valores muito próximos. A diferença média entre FCF e as demais políticas é cerca de 18%. O aumento ocorrido na política FCF foi provocado pela diminuição do tempo médio de resposta dos usuários que possuem conexões rápidas. Com menores tempos de resposta, estes usuários fazem um número maior de requisições, o que contribui para elevar o tamanho médio da *cwnd*. Com maiores janelas de congestionamento, temos uma melhor utilização da capacidade de transmissão da Internet.

A redução do tamanho médio da *cwnd* observado para as políticas FIFO e SRPT ocorreu pelo seguinte motivo. Quando a carga de serviço é baixa (entre 70 e 300 UEs), o tempo de resposta visto pelo cliente é composto basicamente pelo tempo de transmissão na Internet, independente da política. Como os usuários de conexões mais velozes têm tempos de transmissão menores, eles recebem respostas mais rápidas e conseqüentemente fazem um maior número de requisições. Por outro lado, quando a

carga atinge níveis elevados (acima de 300 UEs), o tempo de resposta visto pelo cliente passa a ser composto, em grande parte, pelo tempo de resposta no servidor, ou seja, o tempo de transmissão passa a ter menor influência no tempo final de resposta. Como nas políticas SRPT e FIFO não há privilégio para usuários de conexões rápidas, quando a carga é elevada, estes usuários passam a ter tempos de resposta muito semelhantes aos demais, devido ao atraso imposto pelo servidor. Assim, um menor número de requisições deste tipo de usuário é realizada (comparando-se situações nas quais o servidor opera com carga reduzida). Com menos requisições de usuários de conexões rápidas, o tamanho médio da *cwnd* tende a diminuir, como visto no gráfico da figura 23.

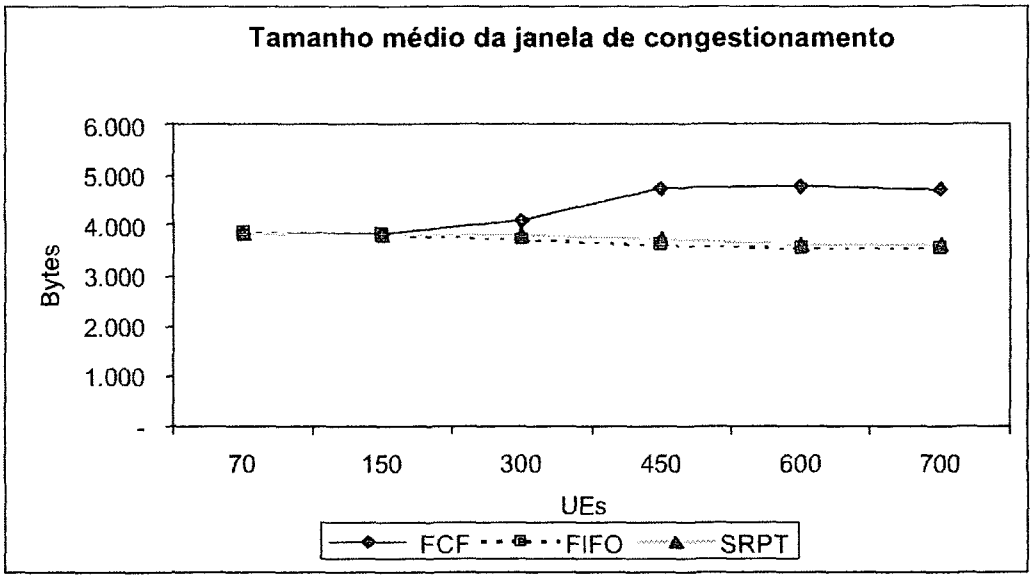

Figura 23 : Tamanho médio da janela de congestionamento.

6.12 Resumo dos Resultados

Em todas as métricas avaliadas a política de escalonamento FIFO apresentou os piores resultados. Aliado aos fatores tradicionais que fazem com que esta política não seja eficiente, outros fatores como a distribuição dos tamanhos de arquivo contribui para o péssimo resultado desta política. A política SRPT comprovou ser imbatível quanto ao tempo médio de resposta no servidor. A política FCF mostrou melhor desempenho nas métricas que envolvem também a Internet, e não apenas o servidor. Deve-se observar que a Internet é um componente fundamental no tempo de resposta visto pelo usuário no ambiente Web. Este tempo de resposta é composto pelo tempo de resposta no servidor, pelo tempo de transmissão na Internet e pela contenção de recursos que ocorre na interação entre servidor Web e Internet. O objetivo da política FCF é justamente

otimizar esta interação, visando um menor tempo final de resposta. A seguir é feito um resumo dos resultados das métricas apresentadas neste capítulo.

Em duas métricas foi observado melhor desempenho da política SRPT, sendo elas o tempo médio de resposta no servidor e o tamanho médio da fila da interface de rede. Em relação ao tempo médio de resposta no servidor, FIFO e FCF apresentaram valores em média 95% e 10% superiores a SRPT. O tamanho médio da fila da interface de rede para FIFO foi 151% superior a SRPT. A política FCF teve um tamanho médio da fila da interface de rede 16% superior a SRPT. Outras métricas como *slowdown*, tamanho médio da fila de disco, tamanho médio da fila de CPU, número médio de requisições por segundo e taxa de serviço em bytes por segundo praticamente não apresentaram diferenças entre as políticas FCF e SRPT. Todavia, o *slowdown* em FIFO foi, em média, 2.127% superior a SRPT. Também na política FIFO o tamanho médio da fila de disco foi 45% superior a SRPT e o tamanho médio da fila de CPU foi 124% superior.

A política FCF apresentou melhor desempenho que SRPT nas seguintes métricas: tempo médio de resposta visto pelo cliente, número médio de requisições pendentes, tempo médio de Internet e tamanho médio da janela de congestionamento. O tempo médio de resposta visto pelo cliente para a política SRPT foi, em média, 10% superior a FCF. A política FIFO apresentou nesta métrica um tempo 53% superior a FCF. O número médio de requisições pendentes na política SRPT foi 27% superior a FCF. FIFO apresentou um resultado muito semelhante a SRPT nesta métrica. O tempo médio de Internet para SRPT ficou em torno de 9% superior a FCF. A política FIFO apresentou um tempo de Internet 25% superior a FCF. Na métrica tamanho médio da janela de congestionamento a política FCF apresentou valores em média 17% superior à política SRPT. FIFO teve um desempenho semelhante à SRPT nesta métrica.

6.13 Validação do experimento

Esta seção apresenta comparações entre os resultados obtidos neste trabalho de dissertação e os resultados descritos na literatura. O objetivo desta comparação é apresentar evidências da consistência e validade dos resultados apresentados.

A carga de trabalho utilizada no nosso experimento foi gerada pelo SURGE [03, 23]. Sua caracterização e resultados são similares aos apresentados no trabalho [16].

No trabalho [16] quando a carga do sistema atinge 95% foi observado que o *slowdown* de FIFO é 20 vezes superior ao do SRPT. Em nosso trabalho, quando o sistema atinge o ponto de utilização máxima (450 UEs), de acordo com o gráfico da figura 17, foi observado um *slowdown* de FIFO 28 vezes superior à SRPT. Embora este valor seja superior ao do trabalho [16], consideramos que pode ser utilizado para validação do experimento.

Com relação ao tempo médio de resposta no servidor não foi observada, nesta dissertação, uma diferença tão expressiva quanto à observada no trabalho [16]. Naquele trabalho, no melhor caso, o tempo de resposta no servidor para FIFO foi três vezes superior a SRPT. No nosso experimento foi duas vezes superior. A explicação para isso é que naquele trabalho a política SRPT foi aplicada também na fila de CPU e de modo preemptivo. No nosso trabalho a política SRPT não foi aplicada na fila de CPU e é não-preemptiva.

No trabalho [24] foi também utilizado o SURGE para avaliar o desempenho do servidor Web Apache. Naquele trabalho foi observado que o tempo médio de resposta ficou próximo de 1,5 segundos quando a carga do SURGE foi de 300 UEs. No nosso experimento observamos um tempo médio de 1,8 segundo com a mesma carga de trabalho. Embora exista uma diferença entre os resultados entendemos que os valores são próximos, o que contribui para validar o modelo adotado neste trabalho.

No trabalho [19] foi implementada a política SRPT na fila da interface de rede utilizando-se o *patch DiffServ* do sistema operacional Linux. Aquele trabalho demonstrou que o tempo de resposta no servidor para a política padrão (FIFO/PS) gira em torno de 10 vezes o da política SRPT para 80% das requisições quando o servidor trabalha com 80% de sua capacidade. Em nosso trabalho, como podemos observar no apêndice 1, o 80º percentil da política FIFO para carga de 300 UE (80% da capacidade) gira em torno de 18 vezes ao da política SRPT. Também foi calculado o tempo médio de resposta para 80% das menores requisições. Neste caso foi constatado um tempo médio para a política FIFO sete vezes superior ao da política SRPT. Acreditamos que as diferenças encontradas entre os resultados desta dissertação e os resultados do trabalho [19] são devidas as diferentes formas nas quais as políticas foram avaliadas. Contudo, observamos que os resultados apresentam semelhanças no aspecto geral e que isso contribui para validar os resultados deste trabalho de dissertação.

No trabalho [18] os autores avaliaram a possível *starvation* dos processos grandes para a política SRPT. A carga de serviço daquele trabalho foi muito semelhante

à utilizada neste trabalho de dissertação. Naquele trabalho foi verificado que quando o sistema opera sobre cargas acima de 90%, a política SRPT apresenta um tempo de resposta para as requisições grandes que não ultrapassa a 5,5 vezes o tempo da política padrão para servidores Web (FIFO/PS). Neste trabalho foi constatado que o maior tempo de resposta no servidor visto para a política SRPT foi de 6.098 segundos e para FIFO foi de 1.125 segundos. Desta forma, na pior situação e com a maior carga testada para o sistema, o tempo de SRPT foi 5,4 vezes superior ao tempo de FIFO. Deve-se ressaltar que, como observado nos trabalhos [16, 18, 19] e neste trabalho, o tempo de resposta no servidor para SRPT é menor que FIFO para mais de 99% das requisições e, em média, 97% inferior à FIFO para todas as requisições.

O modelo adotado para a distribuição do RTT gerou medidas semelhantes às apresentadas no artigo [02], no qual, para 5262 hosts servidores, observou-se uma média de 241 ms, uma mediana de 125 ms e um desvio padrão de 435 ms para o RTT. Neste trabalho foram observadas, a partir dos valores de entrada definidos pela tabela 4, as seguintes medidas para o RTT: média de 262 ms, mediana de 132 ms e desvio padrão de 350 ms. A maior diferença foi do desvio padrão, porém, deve-se observar que um desvio padrão menor desfavorece os nossos resultados. Portanto, a medida é válida e conservativa, isto é, com maior desvio padrão para o RTT poderemos obter melhores resultados.

A modelagem da Internet neste trabalho apresentou resultados coerentes também com o trabalho [46], onde foi demonstrado que o tempo médio de duração de requisições HTTP fica entre 2 e 4 segundos. Neste trabalho observamos um tempo médio de resposta no cliente de 4,48 segundos. Deve-se observar que grande parte deste tempo é devido à demora no servidor que, para quase metade das cargas, apresentou situações de sobrecarga. O custo inerente da Internet, calculado com base apenas no *bandwidth*, RTT, congestionamento e perdas de pacotes ficou em 2,12 segundos, coerente portanto, com o trabalho [46].

No experimento realizado no trabalho [20] foi observado que o tamanho médio da janela de congestionamento foi de 10,43 MSS. Aquele experimento foi realizado em uma rede local onde não foi considerada a possibilidade de perda de pacotes. Na simulação realizada neste trabalho o tamanho médio da janela de congestionamento foi de 6,91 MSS. A diferença observada é decorrente da utilização, neste último caso, dos parâmetros de uma WAN e não de uma rede local. Outro fator que influenciou esta diferença foi o valor da variável *MaxKeepAliveRequests*. No trabalho [20] ela foi

configurada para 150 e nesta simulação para 100, de acordo com a documentação da versão 1.3.14 do servidor Web Apache. Não foram localizados outros trabalhos que permitissem comparar os resultados sobre o tamanho médio da janela de congestionamento.

7. Conclusões e Trabalhos Futuros

7.1 Conclusões

A necessidade de interação com a Internet e as características da carga de serviço tornam o servidor Web um ambiente computacional ímpar. A variabilidade das condições de conectividade observadas na Internet, aliada às características dos protocolos TCP e HTTP, afetam o desempenho do servidor. Considerar as características da carga de serviço e das condições de conectividade trás ganhos de desempenho no tempo de resposta visto pelo cliente, conforme foi mostrado neste trabalho através da política FCF (*Fastest Connection First*). Nestas condições, a política proposta demonstrou ser uma opção interessante para garantir um melhor desempenho de servidores Web.

Com a política FCF observamos uma redução do tempo médio de resposta para o usuário, sem penalizar excessivamente as requisições grandes e as requisições feitas através de conexões lentas. Ficou comprovado que o benefício dado aos usuários com conexões velozes e às requisições menores não traz demora excessiva aos demais usuários e requisições. Observamos através do *slowdown* que o tempo de resposta torna-se mais justo comparativamente ao tamanho da requisição. Priorizando-se conexões mais rápidas no servidor estamos dando um tratamento compatível com o recebido na Internet. Os usuários passam a esperar tempos compatíveis com suas condições de conectividade e os recursos do servidor não são desperdiçados com requisições de conexões pouco eficientes na Internet. Observamos que o tempo de transmissão na Internet pode ser reduzido, evitando que requisições de conexões mais velozes deixem de utilizar a largura de banda disponível devido a atrasos no servidor. Vimos um aumento no tamanho médio da janela de congestionamento o que significa uma melhor utilização da capacidade de transmissão da Internet. Reduções significativas foram observadas no número de requisições pendentes, o que abre espaço para novos estudos sobre a influência que estas reduções trazem no desempenho do servidor.

Não foi observada grande melhora no *throughput* do sistema considerando o número de requisições por segundo e bytes por segundo. Esta melhora menos expressiva deveu-se em parte à dificuldade de simular fatores como a redução nas trocas de contexto e *cache* de disco e, também, pelo fato de que o *throughput* de um servidor Web depende da resposta do sistema como um todo incluindo servidor, Internet e

usuários. Observou-se também, através da comparação do tamanho das filas de disco e interface de rede, que quanto maior a variabilidade da carga de serviço, maior a vantagem da política SRPT no tamanho das filas.

O objetivo principal da política FCF foi melhorar o tempo de resposta para usuários Web através da otimização da interação entre o servidor Web e a Internet. Esta tentativa de otimização foi buscada atribuindo-se mais recursos no servidor às requisições que possuem mais recursos também na Internet, ou seja, podem ser transferidas mais rapidamente pela rede. É correto afirmar que o funcionamento do protocolo TCP implicitamente atribui mais prioridade às requisições de conexões cujas mensagens de confirmação (ACKs) cheguem mais rapidademente no servidor. Contudo, esta prioridade dada pelo TCP ocorre apenas na fila de mensagens de controle do TCP. As demais filas do sistema não atribuem prioridade nenhuma às requisições de conexões mais rápidas. Além disso, uma vez que a requisição possui uma mensagem na fila do TCP, nada garante que seu processamento será prioritário se o tamanho desta fila for grande. Deste modo, a política FCF visa garantir, junto com o TCP, que as conexões rápidas recebam maior prioridade em todas as filas do servidor.

Um fator que dificulta uma melhor avaliação da política FCF é que o modelo de simulação adotado, embora seja baseado em diversos trabalhos [09, 17, 19 e 26], não é suficientemente detalhado para simular todos os aspectos da interação entre o servidor e a Internet. Não foram modeladas as filas dos protocolos TCP e IP. Além disso, as requisições são processadas no disco e na rede em blocos de 16 Kbytes. Desta forma, devido às características da carga Web, quase 85% das requisições são processadas em um único ciclo, distorcendo um pouco a avaliação da interação entre servidor e Internet. Para simular a interação exata do servidor com a Internet, além das filas já modeladas, seria necessário modelar também as filas dos protocolos TCP e IP. Seria necessário realizar o processamento individual de cada segmento e datagrama, o que é muito difícil. Apesar destes inconvenientes, este trabalho apresenta evidências de que priorizar requisições pela velocidade de conexão pode melhorar o tempo de resposta dos usuários Web. Acreditamos que uma avaliação inquestionável da política somente possa ser realizada mediante uma implementação real. Contudo, isto vai além do tempo disponível para este trabalho. Parte dos resultados apresentados nesta dissertação foram publicados no artigo [52].

7.2 Trabalhos Futuros

Uma continuação importante deste trabalho é a implementação da política em um ambiente operacional real tal como a composição Linux mais Apache. Para alterar a ordem de processamento na fila da interface de rede poderia ser utilizado o *Patch DiffServ* [21]. Com este *patch* é possível implementar 16 filas de prioridade (0 a 15). As informações necessárias para definir as prioridades podem ser obtidas do TCB (*Transmission Control Block*), campos *tcb_cwnd, tcb_swindow* e *tcb_srt* [27]. Seria necessária também a alteração do sistema operacional e dos protocolos TCP e IP. Como gerador de carga poderia ser utilizado o SURGE sem necessidade de nenhuma alteração.

A implementação real da política FCF permitiria avaliar o efeito do menor número de requisições pendentes, que provoca a redução do número de trocas de contexto, das estruturas de controle e da quantidade de memória utilizada. Também seria possível avaliar o efeito do *cache* de disco. Poderia ser buscado melhor ajuste da política variando-se o peso dado ao tamanho da requisição e à velocidade de conexão. Uma melhora significativa seria verificar a possibilidade de aplicar a política FCF na fila de CPU tão logo o servidor tenha recebido o pedido da requisição HTTP. Na simulação realizada neste trabalho, a fila de CPU foi escalonada pela política padrão de servidores Web. Esta decisão foi tomada com base em outros trabalhos de simulação e tendo em vista que nas fases iniciais de processamento de uma requisição HTTP é impossível dispor das informações necessárias ao escalonamento. Contudo, em uma implementação real, seria interessante utilizar o escalonamento normal no processamento da fase inicial e, assim que as informações de escalonamento estivessem disponíveis, aplicar a FCF também na fila de CPU.

Na implementação seria possível verificar qual o efeito, na Internet e no servidor, da redução no tempo de resposta dos usuários com conexões mais rápidas e o conseqüente aumento do número de requisições destes usuários. Contudo, o grande problema para avaliar esta política em um ambiente real é a dificuldade de simular a diversidade das condições de conectividade da Internet. A grande maioria dos trabalhos sobre avaliação de desempenho em servidores Web realiza as experimentações em redes locais. No caso da política FCF é impossível avaliar sua eficiência em um ambiente que não apresente a diversidade das condições de conectividade observadas na Internet. Alguns trabalhos [34, 42] alertaram para este problema, porém, suas propostas estão

longe de representar tal diversidade. Testar a política utilizando usuários reais da Internet pode ser uma alternativa, embora seja de alto custo e não permita reproduzir duas ou mais vezes as mesmas condições de conectividade para que duas ou mais políticas sejam comparadas.

Uma questão que deve ser considerada na implementação é o custo do algoritmo de escalonamento. Este custo será composto pelo tempo necessário para acessar os parâmetros da prioridade mais o tempo de escolha nas filas. Como existe a possibilidade de aplicar o escalonamento em mais de uma fila do sistema, deve ser avaliada a vantagem e o custo de processamento em cada uma destas filas. É possível que em algumas destas filas o cálculo para definição da prioridade deva ser simplificado para evitar que o *overhead* sobreponha a vantagem da política de escalonamento. De qualquer modo, o custo da política não foi avaliado e um tratamento especial deve ser dado a ele durante a implementação.

A política FCF, diferentemente da SRPT, utiliza parâmetros dinâmicos para fazer o escalonamento, isto quer dizer que uma mesma requisição pode ter prioridades diferentes no decorrer de sua execução. Este dinamismo é dado pelas variações das condições de conectividade, mais precisamente alterações no tamanho da janela de congestionamento, na janela do receptor e na estimativa do RTT mantida pelo TCP. O tamanho da janela de congestionamento modifica-se em função da fase da conexão e em função das perdas de segmentos. O RTT varia em função do tráfego e congestionamento na interligação com o usuário. A janela do receptor varia em função da disponibilidade de espaço no *buffer* do *host* do usuário. Desta forma, a política de escalonamento deve ajustar dinamicamente a prioridade da requisição. Um fator que deve ser tratado na implementação é a percepção retardada desta alteração pelo algoritmo de escalonamento. Como os parâmetros citados variam muito, é possível que a política utilize valores ultrapassados para calcular a prioridade. Por outro lado, acreditamos que, de um modo geral, estes parâmetros mesmo que utilizados com algum atraso, possam representar a condição média de conectividade da conexão. A política não precisa ser sensível o bastante para capturar pequenas variações de conectividade, mas sim, o suficiente para capturar a capacidade média de transmissão de cada conexão. Em linhas gerais, acreditamos que avaliar a política FCF em um ambiente real seja um trabalho desafiador mas que pode trazer ótimos resultados.

REFERÊNCIAS BIBLIOGRÁFICAS

[01] ALMEIDA, J.; DABU, M.; MANIKUTTY, A.; CAO, P. **Providing Differentiated Levels of Service in Web Content Hosting.** In Proceedings of Workshop on Internet Server Performance, Madison, Wisconsin, June 1998.

[02] CROVELLA, M. E.; CARTER, R. **Dynamic Server Selection in the Internet**. In Proceedings of HPCS'95: Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems, August 1995.

[03] BARFORD, P.; CROVELLA, M. **Generating Representative Web Workloads for Network and Server Performance Evaluation**. In Proceedings of ACM SIGMETRICS'98: International Conference on Measurement and Modeling of Computer System, Madison, July 1998.

[04] ALMEIDA, V. A. F.; ALMEIDA, J; YATES, D. **WebMonitor: a Tool for Measuring World-Wide Web Server Performance**. In Proceedings of Seventh IFIP Conference on High Performance Networking, White Plains, NY, April 1997.

[05] DILLEY, J.; FRIEDRICH, R.; JIN, T.; ROLIA, J. **Measurement Tools and Modeling Techniques for Evaluating Web Server Performance**. In Proceedings of 9th Int. Conf. on Modeling Techniques and Tools, vol. 1245 of Lecture Notes in Computer Science, p. 155-168. Springer-Verlag, 1997.

[06] SPERO, S. E. **Analysis of HTTP Performance Problems**. Disponível em: <http://www.ibiblio.org/mdma-release/http-prob.html> Acesso em: 30 out. 2000.

[07] PADMANABHAN, V. N.; MOGUL, J. C. **Improving HTTP Latency**. In Proceedings of Computer Networks and ISDN Systems, v.28, p. 25-35, December 1995. Disponível em: <http://www.ncsa.uiuc.edu/SDG/IT94/proceedings/Dday/mogul/HTTPLatency.htm ▷ Acesso em: 20 out. 2000.

[08] MURTA, C. D.; ALMEIDA, J.; ALMEIDA, V. A. F. **Performance Analysis of a WWW Server**. In Proceedings of 22nd International Conference on Technology Management and Performance Evaluation of Enterprise-Wide Information Systems, sponsored by the Computer Measurement Group, San Diego, California, December 8-13, 1996.

[09] CHERKASOVA, L. **Scheduling Strategy to Improve Response Time for Web Applications**. In Proceedings of High-performance Computing and Networking: International Conference and Exhibition, p. 305-314, 1998.

[10] Tenth WWW User Survey (Conducted October 1998), Graphics, Visualization & Usability (GVU) Center at Georgia Tech. Disponível em: <http://www.gvu.gatech.edu/user_surveys> Acesso em: 15 set. 2000.

[11] RUBARTH-LAY, J. **Keeping the 400lb. Gorilla at Bay: Optimizing Web Performance.** For LIS385T.6, Electronic Distribution of Organizational Information, Spring 1996.

[12] LUI, B.; ABDULLA, G.; JOHNSON, T.; FOX, E. **Web Response Time and Proxy Caching**. In Proceedings of WEBNET'98: World Conference on the WWW and Internet, Orlando, FL, November 1998.

[13] MURTA, C. D. **Modelo de Particionamento de Espaço para Caches da World Wide Web**. Tese de Doutorado. Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, agosto de 1999.

[14] ACHARYA, A.; SALTZ, J. **A Study of Internet Round-Trip Delay**. Technical Report CS-TR-3736, Department of Computer Science, University of Maryland, USA, December 1996.
Disponível em: <http://www.cs.umd.edu/~acha/papers/latency-tr.html> Acesso em: 30 nov. 2000.

[15] CARDWELL, N.; SAVAGE, S.; ANDERSON, T. **Modeling the Performance of Short TCP Connections.** Technical Report, Department of Computer Science and Engineering, Univ. of Washington, November 1998.

[16] HARCHOL-BALTER, M.; CROVELLA, M.; PARK, S. **The Case for SRPT Scheduling in Web Servers**. Technical Report MIT-LCS-TR-767, MIT Lab for Computer Science, October 1998.

[17] CROVELLA, M.; FRANGIOSO, B.; HARCHOL-BALTER, M. **Connection Scheduling in Web Servers**. In Proceedings of USITS '99: USENIX Symposium on Internet Technologies and Systems, p. 243-254, Boulder, Colorado, October 1999.

[18] BANSAL, N.; HARCHOL-BALTER, M. **Analysis of SRPT Scheduling: Investigating Unfairness**. In Proceedings of ACM SIGMETRICS'2001: International Conference on Measurement and Modeling of Computer Systems, June 2001.

[19] HARCHOL-BALTER, M.; BANSAL, N.; SCHROEDER, B.; AGRAWAL, M. **Implementation of SRPT Scheduling in Web Servers.** Technical Report number CMU-CS-00-170. Carnegie Mellon School of Computer Science, October 2000.

[20] BARFORD, P; CROVELLA, M. **A Performance Evaluation of HyperText Transfer Protocols.** In Proceedings of ACM SIGMETRICS'99: International Conference on Measurement and Modeling of Computer Systems. p. 188-197, Atlanta, Georgia, May 1999.

[21] ALMESBERGER, W.; HADI, J.; KUZNETSOV, A. **Differentiated Services on Linux**. In Proceedings of GLOBECOM'99: Global Telecommunications Conference, December 1999.

[22] BALAKRISHNAN, H.; PADMANABHAN, V. N.; SESHAN, S.; STEMM, M.; KATZ, R. **TCP Behavior of a Busy Internet Server: Analysis and Improvements.** In Proceedings of IEEE INFOCOM. Conference on Computer Communications. p. 252-262, March 1998.

[23] BARFORD, P.; CROVELLA, M. **An Architecture for a WWW Workload Generator.** In Wide Web Consortium Workshop on Workload Characterization, October 1997.

[24] BESTAVROS, A.; KATAGAI, N.; LONDOÑO, J. **Admission Control and Scheduling for High-Performance WWW Servers.** Technical report BUCS-TR-97-015, Boston University, Computer Science Department, August 1997.

[25] CONWAY, R. ; MAXWELL, W. ; MILLER, L. **Theory of Scheduling**. Addison-Wesley Publishing Company, 1967.

[26] DRUSCHEL, P.; BANGA, G. **Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems**. In Proceedings of OSDI'96: Second Symposium on Operating Systems Design and Implementation, October 1996.

[27] COMER, D; STEVENS, D. **Interligação em Rede com TCP/IP: Projeto, Implementação e Detalhes Internos.** Vol. 2. Rio de Janeiro. Campus, 1999.

[28] CROVELLA, M.; BESTAVROS, A. **Self-similarity in World Wide Web Traffic: Evidence and Possible Causes**. In Proceedings of ACM SIGMETRICS'96: International Conference on Measurement and Modeling of Computer Systems, Philadelphia, PA, May 1996.

[29] BARFORD, P.; CROVELLA, M.; BESTAVROS, A.; BRADLEY, A. **Changes in Web Client Access Patterns: Characteristics and Caching Implications**. In Proceedings of World Wide Web: Special Issue on Characterization and Performance Evaluation, volume 12, p. 15-18, 1999.

[30] COMER, D; STEVENS, D. **Internetworking with TCP/IP.** Vol. 1. New Jersey. Prentice Hall, 1995.

[31] ARLITT, M.; WILLIAMSON, C. **Web Server Workload Characterization: The Search for Invariants**. In Proceedings of ACM SIGMETRICS'96: International Conference on Measurement and Modeling of Computer Systems, Philadelphia, PA, May 1996.

[32] EDWARD, S. S.; SUTARIA, J. **A Study of the Effects of Context Switching and Caching on HTTP Server Performance**. Disponível em: <http://www.eecs.harvard.edu/stuart/Tarantula/FirstPaper.html> Acesso em: 12 abr. 2001.

[33] TANENBAUM, A. S. **Computer Networks**. 3. Ed. New Jersey. Prentice Hall, 1996.

[34] BANGA, G.; DRUSCHEL, P. **Measuring the Capacity of a Web Server Under**

Realistic Loads. In World Wide Web Journal (Special Issue on World Wide Web Characterization and Performance Evaluation), 1999.

[35] ALMEIDA, V. A. F; BESTAVROS, A.; CROVELLA, M.; OLIVEIRA, A. **Characterizing Reference Locality in the WWW**. In Proceedings of PDIS'96: The IEEE on Parallel and Distributed Information Systems, Miami Beach, FL, December 1996.

[36] TANENBAUM, A. S. ***Sistemas Operacionais Modernos***. Rio de Janeiro-RJ. Prentice Hall do Brasil, 1995. p. 21, 40 - 50.

[37] SHAY, W. A. **Introduction to Operating Systems**. HarperCollins College Publishers, 1993.

[38] CARTER, R. L.; CROVELLA, M. **Measuring Bottleneck Link Speed in Packet-Switched Networks.** In Proceedings of PERFORMANCE'96, the International Conference on Performance Theory, Measurement and Evaluation of Computer and Communication Systems, Lausanne, Switzarland, October 1996.

[39] CARTER, R. L.; CROVELLA, M. **Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks.** In Proceedings of INFOCOM '97. Technical report TR-96-007, Boston University Computer Science Department. March 18, 1996.

[40] BARFORD, P.; CROVELLA, M. **Critical Path Analysis of TCP Transactions.** In Proceedings of ACM SIGCOMM'2001: Special Interest Group on Computer Communication, Stockholm, Sweden, September 2000.
Disponível em: < http://www.acm.org/pubs/citations/proceedings/comm/347337/p127-barford/>
Acesso em: 30 jan. 2001.

[41] PERIN FILHO, C. **Introdução à Simulação de Sistemas**. 1. Ed. Campinas-SP: Editora da UNICAMP – Campinas, 1995. p. 13 - 26.

[42] BARFORD, P.; CROVELLA, M. **Measuring Web Performance in the Wide Area**. In Proceedings of Performance Evaluation Review, August, 1999.

[43] FLOYD, S.; PAXSON, V. **Why We Don't Know How to Simulate the Internet**. In Proceedings of Winter Simulation Conference, Atlanta, GA, December 1997.

[44] FALOUTSOS, M.; FALOUTSOS, P.; FALOUTSOS, C. **On Power-Law Relationships of the Internet Topology**. In Proceedings of ACM SIGCOMM '99: Special Interest Group on Computer Communication, p. 251-262, August 1999.

[45] KRISHNAMURTHY, B.; WILLS, C. E. **Analyzing Factors That Influence End-to-End Web Performance**. In Proceedings of Ninth International World Wide Web Conference, Amsterdam, Netherlands, April 2000.

[46] DRUSCHEL, P.; BANGA, G.; MOGUL, J. C. **A Scalabel and Explicit Event Delivery Mechanism for UNIX**. In Proceedings of USENIX Annual Technical

Conference, Monterey, California, June 6-11, 1999.

[47] ROUSSKOV, D. We. **The Third Cache-Off, The Official Report.** October 11, 2000. Disponível em: <http://www.measurement-factory.com/results/public/cacheoff/N03/report.by-meas.html/> Acesso em: 25 jul. 2001.

[48] KLEINROCK, Leonard. **Queueing Systems**. Volume 2: Computer Applications. John Wiley and Sons, 1976.

[49] BARFORD, P. **Modeling, Measurement and Performance of World Wide Web Transactions.** Doctoral Dissertation. Boston University, December, 2000.

[50] MENASCE, D. A.; ALMEIDA, V. A. F. **Capacity Planning for Web Performance: Metrics, Models, and Methods**. Prentice-Hall. Englewood Cliffs, NJ, 1998.

[51] ALLEN, A. O. **Probability, Statistics, and Queueing Theory with Computer Science Applications**. Academic Press, 2ª Edition, 1990.

[52] Murta, C. D.; Corlassoli, T. P.; **Política de escalonamento baseada na conexão para Servidores Web.** WSCAD'2001: II Workshop em Sistemas Computacionais de Alto Desempenho em conjunto com o SBAC-PAD'2001, Pirenópolis - GO, Set. 2001.

APÊNDICE 1 - PERCENTIS DIVIDIDOS EM FUNÇÃO DA CARGA E DA POLÍTICA

		Slowdown			Resposta servidor			Resposta cliente			Tempo Internet		
Política	UE	80º	90º	99º	80º	90º	99º	80º	90º	99º	80º	90º	99º
FCF	70	1,35	3,75	58,77	61	140	1.098	1.758	3.409	24.060	1.657	3.285	23.735
FCF	150	2,09	5,78	74,71	76	181	1.738	1.842	3.588	25.907	1.717	3.410	24.673
FCF	300	3,71	8,66	92,05	108	256	2.726	1.770	3.450	27.048	1.602	3.209	24.815
FCF	450	4,42	10,42	90,58	126	311	3.823	1.351	2.652	21.415	1.185	2.430	19.651
FCF	600	4,93	11,66	105,57	128	316	4.420	1.323	2.642	23.182	1.156	2.418	20.171
FCF	700	4,63	11,13	105,66	129	322	4.955	1.388	2.771	25.867	1.210	2.525	22.331
FIFO	70	1,55	5,58	104,01	67	173	1.985	1.836	3.486	23.685	1.662	3.278	23.516
FIFO	150	5,52	25,94	398,27	152	522	6.450	2.131	4.514	25.991	1.747	3.525	25.175
FIFO	300	99,27	239,36	910,64	1.933	4.198	24.116	4.236	7.585	37.090	1.979	4.308	33.145
FIFO	450	257,60	446,25	1.380,20	5.551	8.169	46.969	7.213	11.307	56.632	2.410	5.877	49.078
FIFO	600	420,03	642,93	1.846,84	8.900	12.655	72.299	10.399	15.493	78.417	2.767	8.197	66.842
FIFO	700	528,29	773,55	2.141,78	11.258	15.688	89.314	12.468	18.424	93.843	2.783	10.063	80.314
SRPT	70	1,28	3,61	52,56	61	139	1.080	1.749	3.389	23.832	1.650	3.263	23.550
SRPT	150	1,90	5,58	63,30	76	180	1.646	1.843	3.604	25.681	1.717	3.424	24.688
SRPT	300	3,31	8,12	77,35	108	253	2.543	1.922	3.773	29.393	1.767	3.548	26.910
SRPT	450	4,18	10,21	89,60	125	305	3.913	1.994	3.940	32.068	1.814	3.668	29.035
SRPT	600	4,26	10,49	101,90	127	312	4.672	2.086	4.157	35.173	1.910	3.864	30.380
SRPT	700	4,35	10,83	106,19	129	319	5.286	2.112	4.233	36.871	1.936	3.914	31.577

APÊNDICE 2 - PERCENTIS QUE SÃO IGUAIS PARA TODAS AS POLÍTICAS E CARGAS

	Percentil 80º	Percentil 90º	Percentil 99º
Tamanho de arquivo em bytes	11.239	31.191	176.172
Custo de CPU em ms	4,69	8,59	36,90
Custo de disco em ms	6,44	12,99	71,50
Custo de interface de rede em ms	8,98	24,94	140,90