MODELAGEM DE TRÁFEGO HTTP: DESENVOLVIMENTO E APLICAÇÃO

Kleber Vieira Cardoso

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

Aprovada por:

Prof. José Ferreira de Rezende, Dr.

Prof. Aloysio de Castro Pinto Pedroza, Dr.

Prof. Julius Cesar Barreto Leite, Ph. D.

RIO DE JANEIRO, RJ - BRASIL

JULHO DE 2002

CARDOSO, KLEBER VIEIRA

Modelagem de Tráfego HTTP: Desenvolvimento e Aplicação [Rio de Janeiro] 2002

XIV, 89 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Elétrica, 2002)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Tráfego HTTP
2. Modelo de Agregado
3. Avaliação de Desempenho

I. COPPE/UFRJ II. Título (série)

Agradecimentos

À Deus, por eu existir e ser feliz.

Aos meus pais Raimundo e Rita, pelo amor e apoio durante toda a vida.

À minha irmã Lilian pela amizade.

Ao Prof. José Rezende, por sua orientação e amizade.

Aos Profs. Aloysio e Julius, pela presença na banca e contribuição à tese.

Ao Saulo, pela ajuda na solução de diversos problemas surgidos durante o desenvolvimento do modelo e pela amizade.

Ao Gmoura, pelo auxílio na implementação do gerador de tráfego sobre a plataforma Linux (usando *sockets*) e pela amizade.

Ao Prof. Otto; à Aline, Artur, Beto, David, Diana, Eric, Fagundes, Márcio, Milena, Pedro, Rubi, Tiago e Valentim pelo apoio e amizade.

Ao Prof. Leão; ao Alexandre, Ana Lúcia, Anatércia, Andreia, Anibal, Archimedes, Bagatelli, Belém, Benar, Benjamin, Bernado, Bernardo, Bia, Bicudo, Biondo, Cristina, Cristina Lavrinha, Cristiane, Celso, Daniel, Darci, Doc, Dona Fátima, Edson, Fabricio, Gardel, Gil, Giuliano, Glauco, Granato, Guto, Helvio, Henrique, Hernani, Igor, Isaac, Ivana, Ivone, Jaime, Josenice, Juliana, Keila, Larissa, Leonardo, LG, Luciano, Luis José, Luis Miguel, Marcial, Márcio Lobo, Marcos, Meire, Myrna, Nathalice, Newton, Oscarina, Paulo, Rafael, Renato, Roberta, Roberta Guinther, Rogério, Rubens, Sidney, Silvia, Sônia, Stone, Suzana, Talles, Tux, Vidal e William pela amizade.

À Bia, Solange e Wilson pelo suporte operacional.

Ao PEE/COPPE, pelas instalações e equipamentos utilizados.

À CAPES, pelo financiamento da pesquisa.

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

MODELAGEM DE TRÁFEGO HTTP: DESENVOLVIMENTO E APLICAÇÃO

Kleber Vieira Cardoso

Julho/2002

Orientador: José Ferreira de Rezende

Programa: Engenharia Elétrica

Esse trabalho apresenta o desenvolvimento, avaliação e uso de um modelo de tráfego HTTP de agregado. O modelo oferece um controle simplificado da carga imposta à rede. Além disso, o modelo permite administrar o número de conexões simultâneas e é capaz de reproduzir um comportamento auto-similar. O modelo é do tipo analítico paramétrico, no qual os parâmetros são descritos através de distribuições de probabilidade. Há um número reduzido de parâmetros, sendo que depois da carga, o mais importante é o tamanho da transferência. De forma geral, o modelo é capaz de representar o comportamento de um conjunto de fluxos TCP, os quais transportam conteúdo HTTP e que são responsáveis por gerar uma carga controlada na rede. Para ilustrar o uso do modelo de tráfego proposto foram elaborados estudos de caso que mostram como aplicá-lo em diferentes tipos de avaliações. Um desses estudos apresenta novos resultados sobre o uso de mecanismos de descarte seletivo do tipo *loss conserving* para diferenciação entre classes de serviço e avaliações sobre a perspectiva de se associar políticas de remoção com mecanismos de gerenciamento de *buffer* de descarte prévio.

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

HTTP TRAFFIC MODELING: DEVELOPMENT AND APPLICATION

Kleber Vieira Cardoso

July/2002

Advisor: José Ferreira de Rezende

Department: Electrical Engineering

This work presents the development, evaluation and use of an aggregated traffic model. The model offers a simplified control of the load that is imposed on the network. Moreover, the model allows managing the number of simultaneous connections and is able to reproduce a self-similar behavior. The model is a parametric analytic one, in which the parameters are described by probability distributions. There is a reduced number of parameters and transfer size is the second more important, because the load parameter is the most one. In a general way, the model is able to represent the behavior of a set of TCP flows, which transport HTTP content and are responsible for the generation of a controlled load in the network. To illustrate the use of the proposed traffic model, it was developed studies of case that show how to apply the model in different types of evaluation. A study of case in special presents new results about the use of selective discard mechanisms that are loss conserving in the differentiation between service classes and in the evaluations about the perspective of the joint use of push-out policies with buffer management mechanisms that do early discard.

Lista de Acrônimos

AIMD :	*Additive Increase, Multiplicative Decrease*;
ASCII:	*American Standard Code for Information Interchange*;
ASP :	*Active Server Page(s)*;
ATM :	*Asynchronous Transfer Mode*;
CAIDA:	*Cooperative Association for Internet Data Analysis*;
DHTML:	*Dynamic Hyper Text Markup Language*;
ECN :	*Explicit Congestion Notification*;
FIFO :	*First In, First Out*;
GPS :	*Generalized Processor Sharing*;
HTML :	*Hyper Text Markup Language*;
HTTP :	*Hyper Text Transfer Protocol*;
MIME :	*Multipurpose Internet Mail Extensions*;
MSS :	*Maximum Segmentation Size*;
NS :	*Network Simulator*;
OTcl :	*Object Tool command language*;
RED :	*Random Early Detection*;
REM :	*Random Exponential Marking*;
RTT :	*Round-Trip Time*;
SACK :	*Selective Acknowledgment*;
TCP :	*Transmission Control Protocol*;
UDP :	*User Datagram Protocol*;
URL :	*Universal Resource Locator*;
WFQ :	*Weighted Fair Queuing*;

WRR : *Weighted Round-Robin*;

WWW : *World Wide Web*.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A INTERNET é uma infra-estrutura de rede que se espalha ao longo do globo terrestre. Sua principal função é oferecer conectividade entre os computadores de milhões de pessoas e, em breve, bilhões de pessoas. Há uma grande variedade de aplicações disponíveis que fazem uso da infra-estrutura da Internet, porém uma delas tem se destacado: a *Web*. Para grande parte dos usuários, Internet e *Web* se tornaram sinônimos. O uso intensivo da *Web*, desde a década passada, a tornou uma *killer application*, ou seja, uma aplicação que é usada significativamente mais que qualquer uma de suas concorrentes, tornando-se dominante em termos de pedidos de serviço e de tráfego gerado. Esse fato, fez com o protocolo HTTP, responsável por transferir o conteúdo *Web* pela Internet, se tornasse o mais representativo em inúmeras medições que têm sido feitas na rede.

O número de serviços e a quantidade de informação disponível atualmente na *Web* motiva a continuidade de uma participação significativa do tráfego HTTP. Além disso, algumas características do protocolo HTTP e da aplicação *Web* têm sido razões de sua intensa utilização:

- a *Web* tem se mostrado um meio apropriado para divulgação de informações ou serviços que se baseiem em textos e gráficos;
- o protocolo HTTP é adequado para transferir diferentes tipos arquivos, desde pequenos aplicações Java, até longos vídeos armazenados, passando por

amostras de áudio, *softwares* etc.;

- a *Web* se transformou em uma espécie de interface universal. O aspecto visual (*look and feel*) é simples, amigável e amplamente disponível em várias plataformas.

De acordo com estatísticas recentes da CAIDA [1], o HTTP representa entre 47% e 69% dos bytes enviados ao longo da Internet. Esses números chamam a atenção por dois motivos. Primeiro, pelos valores em si, que ilustram o que foi dito anteriormente. Segundo, pela ampla variação nos resultados apresentados. Essa variação significativa no valor médio se deve à heterogeneidade da Internet em termos de perfis de tráfego nos pontos onde são realizadas as medições. Dito de outra forma, há variações significativas nas amostras de tráfego coletadas o que torna inapropriadas determinadas simplificações como escolha de um único valor médio para descrição de um tráfego. Dessa forma é cada vez mais difícil realizar afirmações a respeito de informações como o tamanho típico do pacote, o atraso médio encontrado ou o percentual que determinado tráfego ocupa na rede. No entanto, esse tipo de informação é importante porque serve como referência para planejamentos, pesquisas e investimentos em tecnologias de telecomunicações e informática.

Apesar do cenário atual da Internet em relação as aplicações e ao tráfego estar de certa forma definido, isso não quer dizer que há uma estabilidade na rede e que não se deve preocupar em avaliar como ela pode mudar no futuro. A Internet tem se mostrado uma rede com características únicas e a possibilidade de mudanças drásticas no perfil do tráfego é uma delas. O melhor exemplo é o do próprio tráfego HTTP, mas há outros como o do MBone e, ainda em menor intensidade, as aplicações ponto-a-ponto (como o Napster). Nesse sentido, a iniciativa Internet 2 [2] é um fator importante. As aplicações e serviços que estão sendo desenvolvidos oferecem uma experiência audiovisual muito superior à oferecida hoje pela *Web*. No entanto, para que alguma das novas aplicações se torne dominante ainda é necessário que a infraestrutura da Internet 2 esteja disponível mundialmente. O tempo necessário para que esse fato ocorra e o ritmo no qual se prevê sua ocorrência, indicam que a *Web* deve permanecer como aplicação representativa por anos.

Dada a ampla utilização da *Web* e conseqüente representatividade do protocolo HTTP no tráfego da Internet, se torna útil compreender e avaliar suas características. Esse trabalho se concentra no estudo desse tráfego. Para realizar essa tarefa, a abordagem escolhida foi o desenvolvimento de um modelo de tráfego HTTP e sua aplicação em alguns estudos de caso. O desenvolvimento de modelos de tráfego HTTP não é um tema novo, e já foi proposto em vários outros estudos [3, 4, 5, 6, 7]. No entanto, como será descrito a seguir, esse trabalho apresenta um modelo com características próprias e mostra em detalhes sua aplicabilidade.

A modelagem de tráfego é uma ferramenta muito importante para estudos envolvendo redes de computadores. Através de modelos de tráfego é possível representar o comportamento de uma aplicação ou agregado de aplicações. Com esse tipo de ferramenta é possível reproduzir artificialmente diferentes situações em uma rede. Dessa forma, com o uso de modelos apropriados, é possível avaliar melhor a infra-estrutura da rede, identificando pontos de congestionamento, enlaces ociosos, qualidade de serviço oferecida às aplicações, entre outros. Quando os modelos de tráfego são utilizados em simulações, é possível não somente avaliar a situação atual da rede, mas também o seu futuro. Como pode ser visto em [8], simulação tem muitas utilidades e vantagens, apesar de possuir também alguns "riscos", sobretudo no que diz respeito a obtenção e interpretação dos resultados.

A maior parte dos modelos de tráfego HTTP desenvolvidos até o momento tem se concentrado em descrever o comportamento de um cliente *Web* [3, 6, 7]. Em geral, os modelos deste tipo se diferenciam pelos parâmetros escolhidos, alterando algumas vezes o grau de detalhamento. Por exemplo, há modelos que não levam em consideração o tempo necessário para processar uma página *Web*. Outros adotam uma sessão "eterna", na qual o usuário sempre desejaria acessar novas páginas, não havendo intervalo entre sessões. No entanto, a maior parte dos trabalhos têm um conjunto comum de parâmetros. A configuração dos valores desses parâmetros também pode variar de um trabalho para outro, porém essas variações, geralmente descritas por distribuições, têm apresentado características semelhantes. Esse tipo de modelo tem se mostrado adequado para situações que exigem um ajuste meticuloso dos parâmetros de um cliente. Em geral, isso ocorre quando se estuda o cliente

Web individualmente ou um pequeno conjunto de clientes. Por outro lado, o número elevado de parâmetros torna esse tipo de modelo complexo, sobretudo quando se usa uma grande quantidade de clientes.

Outro tipo de modelo HTTP é o de agregado, que consiste em representar o comportamento de um conjunto de clientes *Web*. Uma quantidade menor de exemplares desse tipo de modelo foi desenvolvida até o momento [4, 5]. Por ser mais simples que o modelo de cliente, o modelo de agregado é útil para uma ampla variedade de avaliações e estudos, como será descrito em detalhes em um capítulo posterior. Além disso, esse tipo de modelo tende a consumir menos recurso computacional durante a realização de simulações, tornando-se útil ao lidar com altos níveis de agregação, nos quais o modelo de cliente é menos eficiente.

Os modelos desenvolvidos até o momento não apresentam uma forma precisa e simples para configurar a carga que se deseja impor à rede, ainda que esse seja um conceito amplamente aplicado. É muito comum, em trabalhos relacionados a avaliação de desempenho em redes de computadores, se querer observar o comportamento da rede ou de algum dos seus componentes sob uma determinada condição de carga. Por exemplo, o comportamento de um mecanismo de fila quando a carga na rede está próxima a 100% ou como o controle de tráfego reaje com a rede a meia carga.

Outra questão importante é que os modelos desenvolvidos até o momento não apresentam exemplos elucidativos de sua utilização e alguns nem mesmo de sua configuração. Esse fato motiva autores a quererem desenvolver os seus próprios modelos, uma vez que se acredita que o tempo para compreender e usar o trabalho de terceiros é superior ao do desenvolvimento de um novo. Esse trabalho trata também de tal problema através da apresentação de estudos de caso, nos quais a utilização e a obtenção de resultados com o modelo são ilustradas.

De forma sucinta, esse trabalho propõe um novo modelo HTTP, baseado no conceito de comportamento agregado e apresenta duas vantagens principais: pequeno número de parâmetros de configuração e um controle simples e preciso da carga gerada pelo modelo. O desenvolvimento e a avaliação do modelo são apresentados em detalhe. Para realizar o estudo do modelo, foi implementado no simulador NS-2

[9] um gerador de tráfego, tendo como base o modelo HTTP de agregado. Ainda utilizando o simulador, foram elaborados estudos de caso de como o modelo pode ser aplicado em alguns estudos dentre os vários possíveis. Um dos estudos de caso apresenta uma avaliação de desempenho de mecanismos de descarte seletivo de pacotes que utilizam técnicas de *push-out* para a diferenciação de serviços na Internet.

Os capítulos a seguir são organizados da seguinte forma. No capítulo 2 é feita uma revisão bibliográfica do protocolo HTTP, das técnicas de coleta de tráfego e dos tipos de modelagem existentes. É apresentado também um comentário breve sobre os principais modelos, suas características e parametrização. O capítulo 3 apresenta o modelo proposto por esse trabalho. Nesse capítulo é mostrado todo o processo de desenvolvimento e avaliação do modelo. No capítulo 4 são mostrados alguns estudos de casos de aplicação do modelo e seus resultados. Nesse capítulo são apresentados novos resultados sobre o uso de mecanismos de descarte seletivo do tipo *loss conserving* para diferenciação entre classes de serviço e avaliações sobre a perspectiva de se associar políticas de remoção com mecanismos de gerenciamento de *buffer* de descarte prévio. Todos os resultados são obtidos utilizando-se o modelo HTTP proposto. O capítulo discute ainda outras situações no qual o modelo pode ser aplicado. O capítulo 5 faz os comentários finais sobre o trabalho e as perspectivas para trabalhos futuros no assunto.

Capítulo 2

Conceitos Básicos

NESTE capítulo é apresentada, inicialmente, uma descrição das principais versões do HTTP e suas características mais importantes para modelagem de tráfego. Em seguida, são mostrados os métodos tradicionais de modelagem e como o tráfego a ser modelado é coletado para análise. São apresentados ainda alguns modelos de tráfego HTTP, tradicionalmente referenciados na literatura. Por fim, é apresentado um resumo das características do protocolo TCP que influenciam diretamente na modelagem do tráfego HTTP.

2.1 Protocolo HTTP

HTTP é o protocolo da camada de aplicação utilizado na *Web* para comunicação entre clientes e servidores. A *Web* pode ser vista como um conjunto de documentos e serviços disponíveis através da Internet. Os clientes são representados, sobretudo, pelos navegadores que são usados para buscar arquivos nos servidores, também chamados sítios (*sites*). Os arquivos *Web* constituem-se de páginas e objetos *Web*. Uma página consiste, basicamente, em texto ASCII que é conhecido como código HTML - incluindo suas múltiplas variações, como DHTML, ASP etc. Um objeto é um arquivo de imagem, áudio, classe Java ou outro, representado através de um componente embutido em uma página. Os conceitos página e objeto *Web* não são formais, uma vez que não há um consenso nessas definições. Por outro lado, a terminologia escolhida é uma das mais utilizadas e é didática.

De acordo com [10], HTTP é um protocolo do nível de aplicação para sistemas de informação hipermídias, colaborativos e distribuídos. HTTP é um protocolo genérico e sem estado (*stateless*), o qual pode ser usado para muitas tarefas além de seu uso para hipertexto. Outras aplicações incluem servidores de nome e sistemas de gerenciamento de objetos distribuídos, através da extensão de seus métodos de requisição, códigos de erro e cabeçalhos. Entre as características importantes do HTTP estão a tipagem e a negociação de representação de dados, permitindo sistemas serem construídos independentemente dos dados a serem transferidos. HTTP tem sido usado pela iniciativa de informação global WWW desde 1990, e é essa a aplicação do HTTP de interesse neste trabalho.

A primeira versão do HTTP, conhecida como HTTP/0.9, era um protocolo simples para transferência de dados não tratados (*raw data*) através da Internet. O HTTP/1.0, definido em [11], melhorou o protocolo permitindo que os dados fossem transferidos no formato de mensagens MIME, contendo meta-informação sobre os dados transferidos e modificadores de semântica requisição/resposta. MIME é uma especificação para formatar mensagens não-ASCII de tal forma que elas possam ser enviadas pela Internet, permitindo aos navegadores exibirem arquivos que não estão no formato HTML, tais como gráfico, áudio e vídeo. No entanto, o HTTP/1.0 não foi projetado para tratar apropriadamente os efeitos de *proxies* hierárquicos, *caching* ou a necessidade de conexões persistentes. Além disso, houve uma proliferação de implementações HTTP/1.0 com diferentes características e uma necessidade, de cada lado de uma comunicação, de identificar quais capacidades estavam disponíveis no extremo oposto. Para atender essas necessidades, o HTTP/1.1 foi desenvolvido. Mais adiante serão apresentados mais detalhes sobre as duas versões do protocolo HTTP mais utilizadas atualmente (HTTP/1.0 e HTTP/1.1) e suas principais características.

Como já foi dito, o HTTP é um protocolo do tipo requisição-resposta projetado para transferir os arquivos *Web*. Cada transferência consiste na requisição de arquivos (páginas ou objetos) pelo cliente ao servidor e a resposta do servidor com os arquivos pedidos ou com notificações de erro. De forma mais precisa, o usuário requisita ao navegador o acesso a um URL ou simplesmente sítio, e uma página é então requisitada ao servidor. Ao receber a página, o navegador a varre em busca de objetos que a compõem, como imagens, sons, classes Java, entre outros e requisita esses objetos ao servidor. O usuário tem então disponível as informações que pediu e *links* disponíveis para outras páginas. Caso o usuário selecione algum *link*, todo o processo anterior se inicia novamente. A figura 2.1 ilustra esse processo através de um diagrama de tempo para troca de mensagens HTTP/TCP.

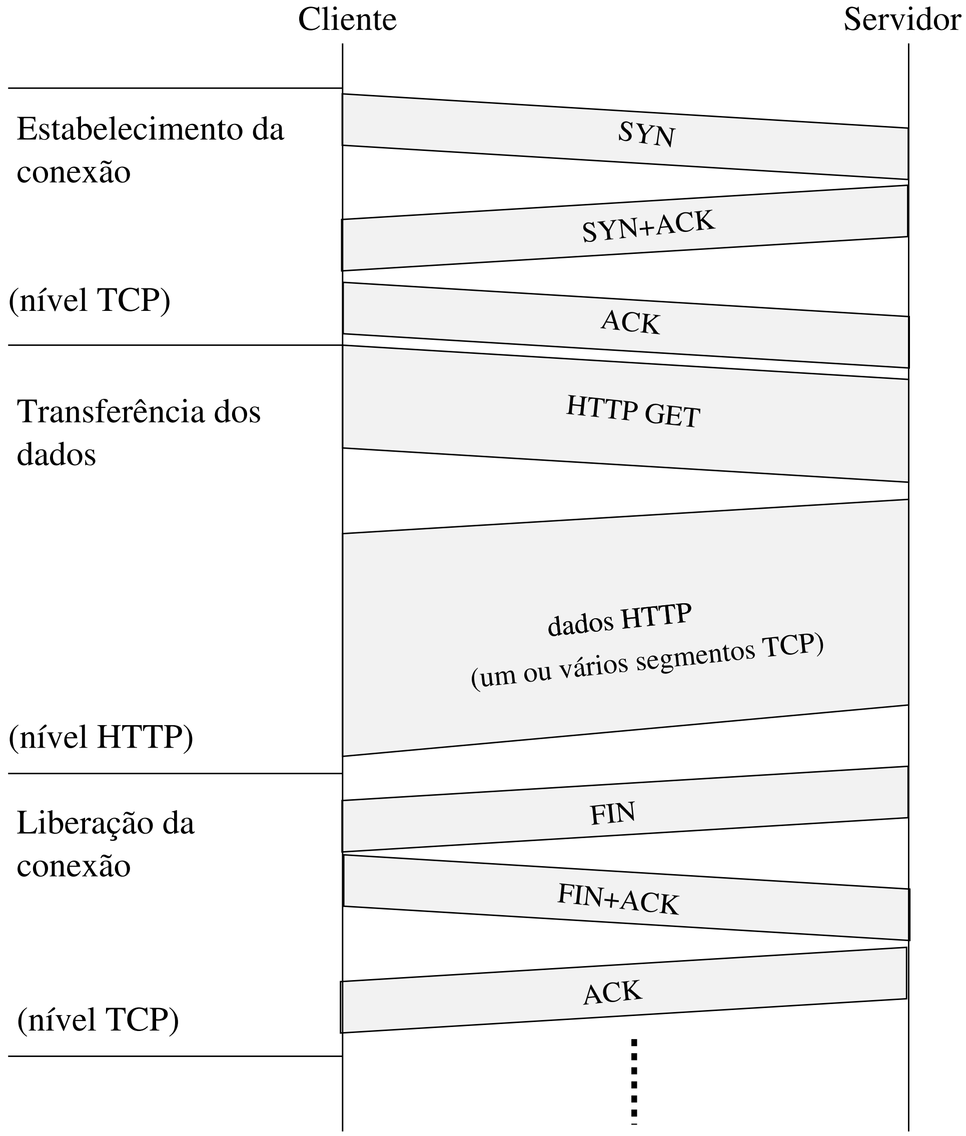


Figura 2.1: Seqüências de mensagens HTTP/TCP ao longo do tempo.

O HTTP utiliza, normalmente, a pilha TCP/IP para se comunicar, sendo que o servidor HTTP tipicamente responde a partir da porta TCP 80. Porém, o HTTP pode ser implementado sobre outros protocolos, desde que o protocolo de transporte seja confiável. Nesse trabalho, somente será considerado o uso do HTTP sobre o TCP, por ser o protocolo de transporte confiável de uso comum na Internet.

2.1.1 Características e Diferenças do HTTP/1.0 e HTTP/1.1

Conforme pode ser visto em [12], a versão 1.1 do protocolo HTTP trouxe várias inovações em relação a versão 1.0, e algumas delas têm influência no desenvolvimento de modelos de tráfego desse protocolo. Na descrição que segue, serão apresentadas apenas as características que influenciam, de forma mais significativa, a modelagem do tráfego HTTP, sendo que mais detalhes podem ser obtidos em [10, 12].

Banda Passante

O desperdício de banda passante pode levar ao aumento da latência, piorando o serviço de transmissão dos dados e agravando eventuais congestionamentos. Essa situação se agrava em enlaces de baixa capacidade. O HTTP/1.1 trata algumas das deficiências do HTTP/1.0 no uso eficiente da banda passante.

No HTTP/1.0 não há como um cliente requisitar apenas parte de uma página ou objeto. Se, por exemplo, um usuário requisita uma página extensa e com vários objetos embutidos, mas lê apenas as primeiras linhas e obtém a informação que procurava, ainda que não tenha feito nenhuma rolagem da tela, toda a página e seus objetos serão buscados, desconsiderando o caso de uma interrupção explícita por parte do usuário. O HTTP/1.1 permite ao cliente requisitar porções de um recurso, através da requisição de uma faixa de *bytes* desejados, extensível para outras faixas (por exemplo, faixa de objetos). Caso a faixa pedida não atenda ao usuário, o cliente pode pedir quantas faixas contíguas adicionais forem necessárias. Voltando ao exemplo anterior, caso o cliente tenha buscado apenas o suficiente para preencher uma tela e o usuário role a tela, uma ou mais requisições são geradas para completar a exibição da página.

Algumas requisições HTTP podem ser arbitrariamente longas, carregando, por exemplo, o conteúdo de formulários. Se o servidor não aceita uma requisição, por exemplo devido a uma falha na autenticação, a banda passante consumida na transmissão da requisição foi desperdiçada. No HTTP/1.0 não há como saber se a requisição será atendida antes de transmiti-la completamente. O HTTP/1.1 inclui um novo código de *status* (*status code*) que permite ao cliente separar o cabeçalho do corpo da requisição, o qual pode ser grande. O cliente primeiro envia o cabeçalho da requisição e espera por uma resposta, se a resposta for um código de erro significa que o cliente não deve enviar o corpo da requisição, pois essa não será atendida. Caso contrário, o cliente pode enviar o corpo da requisição para o servidor tratá-la.

Uma forma bem conhecida de economizar banda passante é através da compressão de dados. Apesar de alguns objetos *Web* terem seus formatos nativos compactados, como imagens e vídeos, outros como texto podem ser comprimidos e reduzirem significativamente a quantidade de *bytes* transmitidos. Apesar do HTTP/1.0 incluir algum suporte para compressão, o protocolo não fornece mecanismos adequados para negociar o uso da compressão ou para distinguir entre compressão fim-a-fim ou salto-a-salto. O HTTP/1.1 oferece novos recursos que lidam com essas deficiências.

Gerenciamento da Conexão de Rede

Como foi dito anteriormente, o TCP é o protocolo de transporte efetivamente em uso pelo HTTP. No HTTP/1.0 cada requisição de página ou objeto é feita usando uma conexão TCP independente e as conexões são serializadas, ou seja, uma nova conexão somente é aberta após a requisição transportada pela conexão anterior ter sido atendida e a conexão encerrada. Logo, cada requisição implica no custo de estabelecer uma nova conexão TCP. Além disso, a maior parte das páginas e objetos tem poucos Kbytes de tamanho. Logo, poucas conexões TCP passam da fase de *slow start*, tendo portanto um baixo desempenho [13, 14]. Para amenizar a serialização, algumas implementações do HTTP/1.0 fazem uso de conexões TCP paralelas. Porém, aumentar o número de conexões simultâneas que um único aplicativo usa, eleva as chances de congestionamentos ocorrerem [15], pois o número de fluxos TCP concorrendo pelo mesmo recurso aumenta. Se esse fato ocorre de forma abrangente e indiscriminada, existe o risco de se ter colapsos de congestionamento [15]. Por essa razão, os navegadores mais populares da Internet limitaram o número de conexões paralelas a 4 ou 6.

Para evitar a sobrecarga causada pela abertura de conexões independentes para cada par requisição/resposta, no HTTP/1.0 foi introduzido o uso de mensagens do tipo *keep-alive*. Essas mensagens mantêm uma conexão TCP aberta entre cliente e servidor durante o tempo que for necessário para transmissão de quantas páginas e objetos o usuário desejar. O HTTP/1.0 não possuía esse recurso documentado originalmente, mas algumas implementações passaram a fazer uso deste. No entanto, não há interoperabilidade entre as mensagens *keep-alive* e eventuais *proxies* intermediários, podendo até levar a problemas de funcionamento dos *proxies* [10]. No HTTP/1.1, foi introduzido o conceito de conexão persistente, que funciona de forma semelhante às mensagens *keep-alive*, porém clientes, servidores e *proxies* estão preparados para lidar com esse tipo de conexão. Na verdade, esses componentes da rede são configurados para considerar que, inicialmente, esse seja o tipo de conexão a ser usado. O protocolo mantém ainda compatibilidade retroativa com o HTTP/1.0 e permite que a conexão seja encerrada a qualquer momento por qualquer uma das extremidades. No HTTP/1.1 o número máximo recomendado de conexões simultâneas é dois. Apesar de alguns problemas observados no uso de conexões persistentes [16], esse tipo de conexão se mostrou uma solução mais eficiente que as conexões independentes e, portanto, no HTTP/1.1 se tornou o padrão.

No HTTP/1.0, apesar de uma conexão ser capaz de transmitir vários pares requisição/resposta, o processo continua sendo serializado. Ou seja, uma nova requisição é gerada somente após a resposta da última requisição ter chegado completamente. Isso faz com que o intervalo mínimo entre o início de cada requisição seja pelo menos um RTT e, na maior parte do tempo, é superior a isso porque as respostas consomem mais que um pacote. O HTTP/1.1 elimina a deficiência da serialização através do *pipelining*. Com esse recurso um cliente pode enviar um número arbitrariamente grande de requisições em uma única conexão TCP antes de receber qualquer resposta. Essa técnica aumenta o desempenho do protocolo HTTP, que passa a fazer um uso melhor do protocolo TCP.

Caching

Informações como conteúdo ao vivo, anúncios freqüentemente atualizados ou bate-papos pela *Web* tem, recentemente, levado o *Web caching* a perdas de eficiência. Ou seja, o número de acertos dos servidores *proxy* tem diminuído independentemente dos algoritmos de *caching* utilizados. Por outro lado, essa técnica ainda é capaz de reduzir a quantidade de informação transmitida ao longo da rede e melhorar a experiência de navegação do usuário. Naturalmente, como conseqüência do *caching* o consumo de banda passante é reduzido, pois se evita transmitir pacotes desnecessários. Com a redução do consumo de banda passante, se reduz indiretamente a latência das interações não submetidas a *caching* através da redução do congestionamento na rede.

O HTTP/1.0 fornece um mecanismo de *caching* simples e que funciona adequadamente em um número significativo de situações, porém ele tem muitas falhas conceituais. O *caching* do HTTP/1.0 não permite servidores ou clientes fornecerem instruções completas e explícitas para os *caches*, logo, ele depende de um conjunto de heurísticas que não foram bem especificadas. Esse fato leva a *caching* incorretos de algumas respostas que não deveriam ter sido guardadas (*cached*), levando a problemas semânticos. Outro problema é que há falhas ao não armazenar (*cache*) algumas respostas que podiam ser armazenadas, levando a perda de desempenho.

O HTTP/1.1 tenta tornar nítidos os conceitos de *caching* e fornecer mecanismos extensíveis e explícitos no protocolo para realizar essa tarefa. Ao mesmo tempo que mantém o projeto básico do HTTP/1.0, o HTTP/1.1 melhora o projeto com novas características e com uma especificação mais cuidadosa das já existentes.

2.2 Modelagem de Tráfego HTTP

Em análise de desempenho em redes de computadores, modelos de tráfego apropriados são ferramentas importantes e influenciam na obtenção de dados úteis. Um modelo pode ser considerado apropriado se ele for capaz de gerar tráfego com características semelhantes às observadas no gerado por uma determinada aplicação que se pretende representar. A maior parte dos trabalhos sobre a aplicação *Web* tem se concentrado em elaborar modelos de clientes [3, 6, 7, 17], os quais tentam descrever o comportamento de um cliente *Web* individualmente. Outra abordagem é modelar o comportamento observado em um agregado de clientes *Web* [4, 5], ou seja, criar um modelo de agregado. Essa taxonomia ainda não é consenso na literatura, mas as definições apresentadas são importantes para a compreensão do restante do texto deste trabalho.

Tanto o modelo de cliente quanto o de agregado apresentam vantagens e desvantagens. O modelo de cliente consegue capturar mais detalhes da aplicação, sendo portanto, uma imitação mais fiel da mesma. Porém, a complexidade de desenvolvimento desse tipo de modelo é maior em relação ao de agregado e em determinadas situações o nível de detalhe oferecido não contribui na avaliação que se deseja realizar. Exemplos disso são estudos relacionados a provisionamento de recursos, comportamento de filas, mecanismos de escalonamento, entre outros. Nesses casos, modelos mais simples são mais apropriados. O modelo de agregado é, geralmente, uma aproximação mais grosseira do tráfego real, pois negligencia mais detalhes. Ainda assim, é uma ferramenta muito útil porque permite, através do controle de poucos parâmetros, simular condições e identificar comportamentos que dificilmente se conseguiria com o modelo de cliente, devido a sua complexidade. Além disso, para reproduzir um grande número de clientes *Web*, modelos de cliente tendem a ter um custo computacional mais elevado que os modelos de agregado.

Nos dois tipos de modelo, um dos fatores mais importantes é a escolha das características da aplicação que se deseja reproduzir. Como uma opção de projeto, o modelo pode ser desenvolvido em torno de tais características. Alguns exemplos dessas características são: carga gerada na rede, dependência de longa duração no tráfego da aplicação, quantidade de clientes *Web* simultâneos, popularidade dos arquivos acessados, fatores humanos como intervalo entre "cliques", tempo de visualização e duração de uma sessão etc.

Independente de ser de cliente ou de agregado, um modelo *Web* utiliza parâmetros para tentar reproduzir determinadas propriedades dessa aplicação. Exemplos desses parâmetros são tamanho do arquivo sendo transferido, intervalo entre páginas, número de objetos por páginas, entre outros. Para descrever esses parâmetros duas abordagens podem ser utilizadas: baseada em amostras de tráfego real ou analítica. Os modelos produzidos a partir dessas abordagens são conhecidos como estruturais [18], pois tentam caracterizar a natureza do tráfego.

O uso de amostras de tráfego real consiste em descrever determinado parâmetro da aplicação através de um conjunto de valores predefinido, coletado a partir de um ambiente de rede real. As vantagens desse método são a facilidade de implementação e a reprodução de um sistema conhecido. Porém, essa abordagem trata o tráfego gerado como uma "caixa-preta", tornando difícil a compreensão do comportamento observado. Além disso, o gerador de tráfego construído a partir desse modelo se torna difícil de ajustar, uma vez que novas condições ou demandas variáveis não são trivialmente configuráveis.

A abordagem analítica consiste no uso de distribuições de probabilidade na descrição de um determinado parâmetro. Uma distribuição de probabilidade descreve como uma seqüência de valores aleatórios se comporta, dado um número significativamente alto de amostras, e uma vez conhecida, permite a geração de novas (e diferentes) seqüências que obedeçam a mesma distribuição. A desvantagem dessa abordagem é a dificuldade encontrada em identificar uma distribuição, e sua configuração, que descreva adequadamente a seqüência de valores aleatórios dos parâmetros da aplicação. Quando não se encontra uma distribuição de probabilidade (e sua respectiva configuração) apropriada para descrever um determinado parâmetro e aproximações não são aceitas, é possível representar a distribuição de probabilidade "desconhecida" através de sua função de distribuição acumulativa e usar o método de transformação inversa para obter os valores. Embora seja capaz de representar distribuições arbitrárias, essa solução exige maior capacidade de armazenamento e é mais lento.

Há ainda uma terceira abordagem, que consiste no uso de processos abstratos conhecidos para tentar capturar somente as propriedades estatísticas do tráfego com independência dos mecanismos subjacentes de geração do tráfego. Essa abordagem é computacionalmente eficiente e, relativamente, simples de implementar. Além disso, ela pode ser útil quando características muito específicas desejam ser estudadas. Por exemplo, a auto-similaridade pode ser reproduzida por um processo Browniano. No entanto, esse tipo de abordagem não leva em consideração fatores importantes para a formação do perfil do tráfego e negligencia elementos como o controle de congestionamento do TCP, que tem influência marcante no tráfego HTTP. Ou seja, o tráfego volta a ser tratado como uma "caixa-preta", embora de uma forma diferente, mas com conseqüências semelhantes. Modelos criados a partir dessa abordagem são conhecidos como "comportamentais" [18].

Nesse trabalho o enfoque é um modelo de tráfego analítico com número reduzido de parâmetros e capaz de reproduzir uma determinada carga na rede. O modelo é chamado de agregado porque tem como objetivo representar a carga que seria gerada por um determinado número de clientes *Web* (não especificado). No próximo capítulo será apresentado em detalhes o modelo de agregado que foi desenvolvido, assim como suas motivações e implicações.

2.2.1 Coleta e tratamento do tráfego HTTP

O desenvolvimento de modelos de tráfego para aplicações interativas exige um estudo do tráfego efetivamente gerado pelas aplicações. Ou seja, para desenvolver um modelo de tráfego *Web*, é importante coletar amostras desse tráfego geradas em ambientes reais e extrair as informações necessárias. A extração de determinadas informações do tráfego *Web* não é trivial e exige o desenvolvimento de heurísticas sofisticadas. A seguir são apresentados os principais métodos utilizados na coleta de tráfego *Web* e as informações de maior interesse para o desenvolvimento de modelos de agregado e cliente.

O método de **registros do servidor** (*server logs*) se baseia na capacidade dos servidores *Web* de guardar informação sobre os arquivos que eles servem. A principal vantagem desse método é a facilidade de se coletar os dados, uma vez que os recursos já estão disponíveis, ou seja, os registros dos servidores. Para um modelo de agregado, esse método pode ser útil, uma vez que as informações sobre vários clientes que acessam determinado servidor estão disponíveis. Porém, se restringe aos acessos a servidores individuais, tornando difícil identificar o comportamento em enlaces que interliguem clientes com múltiplos servidores. Ou seja, o modelo pode se tornar excessivamente restrito. Outra desvantagem desse método é que os registros do servidor não contém os cabeçalhos HTTP, que representam uma sobrecarga do protocolo. No caso específico de modelo de cliente, esse método não permite, facilmente, capturar os padrões de acesso entre múltiplos servidores. Isto ocorre porque é difícil identificar os diferentes servidores que são eventualmente acessados por um usuário.

Um segundo método é o de **registros de clientes** (*client logs*) e consiste em coletar informações a partir de navegadores modificados, capazes de guardar dados do comportamento dos usuários. Esse método foi utilizado no passado [19, 20], mas é de difícil implementação atualmente porque exige que um número suficientemente grande de navegadores com capacidade de coletar dados esteja disponível para usuários *Web* típicos. Para o desenvolvimento de modelos de cliente, esse método é muito útil, pois permite que uma grande quantidade de detalhes da aplicação seja capturada. Para modelos de agregado, surge a dificuldade de identificar o comportamento de uma agregação de clientes.

Outro método conhecido é o **registros de pacotes** (*packet traces*). Esse método tem sido o mais utilizado em trabalhos recentes e consiste em coletar pacotes ou parte deles a partir de uma sub-rede que transporta tráfego HTTP. As coletas tem variado desde redes locais Ethernet até espinhas dorsais, como as realizadas por alguns projetos de destaque [21, 22] que capturam grandes volumes de informação. Esse método apresenta algumas dificuldades na identificação de parâmetros como número de objetos por página efetivamente transmitidos e intervalo entre páginas. Porém, várias técnicas (ou heurísticas) foram desenvolvidas e ferramentas de domínio público como HTML-REDUCE [4] e BLT [23] têm apresentado resultados satisfatórios e auxiliado no desenvolvimento de modelos.

Por fim o método de **registros de *proxies*** (*proxies* logs), o qual permite obter informações de forma semelhante à do método anterior. Uma desvantagem do método é que um *proxy*, normalmente, atende uma única rede ou um pequeno grupo de redes locais, enquanto o método anterior pode coletar informações em uma rede de trânsito entre várias redes. Logo, a heterogeneidade de clientes e servidores do método anterior é maior.

Uma vez que o tráfego foi coletado, tem início o processo de identificar e contabilizar os parâmetros de interesse. Esses parâmetros variam de um modelo para outro. Alguns aparecem em vários modelos, como o tamanho de uma página ou objeto *Web*, outros são muito específicos de um determinado modelo como o número de páginas consecutivas. Alguns parâmetros são medidos com facilidade, outros exigem heurísticas sofisticadas. Segue uma breve descrição dos parâmetros mais comumentes usados e alguns exemplos de como descrevê-los estatisticamente.

- Tamanho de página ou objeto - alguns trabalhos apresentam distinção entre a página requisitada pelo usuário e os demais componentes da página como imagens, scripts etc. No entanto, a maioria dos autores tratam a página como mais um objeto ou, simplesmente, o objeto principal. Esse é o parâmetro mais medido e avaliado na literatura com relação ao tráfego HTTP. O tamanho da página/objeto influencia tanto HTTP/1.0, quanto o HTTP/1.1 porque tem uma relação direta com o funcionamento do controle de congestionamento do TCP. Além disso, alguns estudos [24, 25, 26] identificam esse parâmetro como um dos responsáveis pelo comportamento auto-similar do tráfego HTTP. A maior parte da medições indicam que os valores desse parâmetro são melhor descritos por uma distribuição de cauda longa. Pareto é a distribuição preferida para configurar os valores desse parâmetro, mas há também trabalhos que utilizam Lognormal ou híbrida (Pareto + Lognormal).

- Número de objetos por página - define a quantidade de elementos que compõem uma página *Web*. Para o HTTP/1.1 esse parâmetro é muito importante porque a quantidade de objetos define a eficiência da conexão persistente e do *pipeline*. Quanto maior o número de objetos em uma única página, melhor tende a ser o desempenho do HTTP/1.1 se comparado ao HTTP/1.0 tradicional. Não há muito consenso nos valores assumidos por esse parâmetro, e diferentes distribuições foram identificadas para descrevê-lo, por exemplo, Gama, Lognormal e Pareto.

- Tempo de visualização - descreve o comportamento típico de um usuário *Web* enquanto está acessando um conteúdo *Web*, ou seja, o tempo consumido ao ler o texto de uma página, preencher um formulário, visualizar um vídeo etc. É um parâmetro difícil de ser avaliado por representar um comportamento humano e, novamente, não há consenso nos resultados. Além disso, é muito difícil falar em valores característicos para esse parâmetro, devido ao fator humano. Há uma uma grande variedade de usuários acessando diferentes tipos de documentos e serviços *Web* e determinar qual conjunto é mais representativo é uma tarefa complexa. Por fim, o comportamento de um determinado usuário pode mudar à medida que varia o conteúdo disponível na *Web*, ou as condições da rede, ou o custo do acesso, entre outros, tornando o parâmetro altamente variável. Algumas distribuições escolhidas têm sido Weibull, Gama e Lognormal.

- Tempo de silêncio - representa o tempo entre sessões ou páginas, isto é, o período entre o recebimento do último objeto até o próximo pedido do usuário. Também é considerado um parâmetro de difícil medição e avaliação, sobretudo quando está definindo o intervalo entre páginas. Nessa situação é possível (e comum) ocorrer tempos nulos, pois o usuário pode gerar uma nova requisição antes da atual ser atendida por completo. Tem sido escolhido para descrever esse parâmetro, a distribuição Pareto ou o processo de chegada de Poisson.

- Tamanho da requisição - apesar da maior parte dos trabalhos arbitrarem um valor para esse parâmetro, alguns autores preferem medi-lo, na tentativa de aumentar a acurácia do modelo. As variações significativas do tamanho da requisição HTTP ocorrem, basicamente, por conta do uso de formulários. Exemplos de distribuições utilizadas são Lognormal e Bimodal.

Além dos parâmetros apresentados existem vários outros que são específicos dos modelos onde foram propostos e cuja influência no tráfego não tem se mostrado significativa. Alguns exemplos são: número de páginas por sessão, tempo para varredura de uma página *Web*, localidade temporal, tempo entre chegadas de objetos, número de páginas em *cache*, entre outros.

A tabela 2.1 mostra as distribuições escolhidas por alguns trabalhos para representar os principais parâmetros descritos anteriormente. Como pode ser observado, o tamanho de página/objeto é o parâmetro mais comum. Isso ocorre, sobretudo, pela influência desse parâmetro em todos os modelos.

Tabela 2.1: Exemplos de configurações de parâmetros.

	Tamanho de página ou objeto	Número de objetos por página	Tempo de visualização	Tempo de silêncio	Tamanho da requisição
[3]	Pareto	-	-	-	Bimodal
[4]	Lognormal	Lognormal	Lognormal	-	Lognormal
[5]	Híbrida	Pareto	Weibull	Pareto	-
[7]	Lognormal	Gama	Weibull	-	Lognormal
[24]	Pareto	-	-	-	-
[26]	Híbrida	-	-	-	-
[27]	Pareto	-	Gama	Poisson	-

2.2.2 Exemplos de modelos de tráfego HTTP

A seguir é apresentada uma breve descrição de alguns modelos de tráfego amplamente citados na literatura. Embora, existam vários outros modelos, eles são, em geral, variações dos apresentados a seguir. Dos cinco trabalhos comentados, três destacam a propriedade de auto-similaridade, a qual será comentada no próximo capítulo.

No trabalho [3] é proposto um modelo HTTP de cliente e realizado um estudo, medição e avaliação de vários parâmetros que inspiraram novos trabalhos. Os parâmetros considerados mais importantes nesse trabalho são: tamanho da requisição, tamanho da resposta (equivalente ao objeto principal), tamanho do documento (semelhante ao número de objetos por páginas) e tempo de pensamento (equivalente ao tempo de visualização). Esse trabalho utilizou a técnica de registros de pacotes para avaliar o tráfego e escolheu parâmetros que não são governados pelo controle de fluxo do TCP e pelos algoritmos de controle de congestionamento. Para descrever os valores assumidos pelos parâmetros, o autor escolheu funções de distribuição e funções de distribuição acumulativa.

Em [7] são apresentadas as dificuldades em identificar com precisão os limites entre páginas quando a técnica de coleta utilizada é a de registros de pacotes. Para contornar o problema os autores propõem um modelo de cliente que faz uso de uma requisição-*Web* na qual uma página ou um conjunto de páginas e seus respectivos objetos são trazidos a partir de uma ação do usuário. Entre os parâmetros utilizados estão o tamanho do objeto, tamanho da requisição, número de objetos por página e tempo de visualização. Os parâmetros são descritos através de distribuições.

No trabalho [6] é proposto um modelo de cliente baseado nos "cliques" dos usuários e na quantidade de informação transferida em cada uma dessas ações. Para coletar as informações que necessita, os autores utilizam uma combinação de registros de pacotes com um *proxy* de servidor X, que permite verificar o tempo dos "cliques". Os autores preferem trabalhar com um conjunto reduzido de parâmetros, a saber: tempo em que ocorrem os "cliques", tempo de silêncio e quantidade de informação transferida, que é uma agregação do tamanho de páginas e objetos trazidos a partir de uma ação do usuário. A descrição dos valores é feita através de distribuições.

Em [5] é apresentado um modelo HTTP de agregado para geração de carga em rede. Os autores propõem o conceito de usuário equivalente, que corresponde à carga gerada por um número conhecido de usuários, para uma determinada configuração da rede. Os autores utilizam navegadores modificados para coletar as informações que necessitam. Entre os parâmetros utilizados estão: tamanho de objeto, tamanho de requisição, número de objetos por página e tempo de visualização. São usadas distribuições para descrever os valores assumidos pelos parâmetros.

Em [4] é mostrado um modelo de agregado para dimensionamento de largura de banda. Os autores também discutem as dificuldades de identificar com precisão os limites entre páginas ao se utilizar a técnica de registros de pacotes e desenvolvem uma ferramenta que sintetiza as heurísticas que propõem. Alguns parâmetros avaliados no trabalho foram: tamanho de página, número de objetos por página e tempo de silêncio. Os autores realizam um estudo estatístico detalhado dos valores dos parâmetros e apresentam algumas opções para as distribuições que os modelam, sugerindo as que mais se adequam.

2.3 Protocolo TCP

Como foi comentado anteriormente, o HTTP utiliza o TCP como protocolo de transporte, o qual exerce influência no comportamento do tráfego e compreendê-lo facilita o processo de modelagem. A maior parte dos modelos de tráfego HTTP evitam utilizar parâmetros que sejam diretamente afetados pelo protocolo de transporte (por exemplo, a taxa de chegada de pacotes), apesar de alguns ainda o fazerem. Por outro lado, ao avaliar o comportamento de um modelo de tráfego HTTP é útil considerar o protocolo de transporte, pois alguns comportamentos são influenciados ou até mesmo gerados por esse protocolo.

Atualmente, há uma variedade de implementações do TCP em uso, como o Reno, NewReno e SACK [28, 29, 30]. Essas implementações apresentam algumas diferenças, no entanto, o comportamento AIMD (que afeta de forma significativa o comportamento do tráfego) é semelhante em todas. Isso se deve às duas principais fases de controle de congestionamento: *slow start* e *congestion avoidance*. Durante essas fases o TCP realiza uma sondagem, em diferentes níveis, da quantidade de banda passante disponível para uma dada conexão. As fases são separadas por um limiar de congestionamento, que é alterado na ocorrência de perdas. Na fase de *slow start*, a janela de congestionamento aumenta de um segmento a cada reconhecimento, e portanto, a vazão dobra a aproximadamente cada RTT. Na fase de *congestion avoidance* a abordagem é mais conservadora e a janela cresce um segmento após todos os reconhecimentos de uma janela terem chegado. Quando uma perda é detectada por temporização a janela volta a ter o tamanho inicial de um segmento e todo processo tem início novamente.

De fato, há muito mais detalhes sobre o controle de congestionamento do que foi apresentado no parágrafo anterior. Porém, ele é suficiente para ilustrar como a vazão do TCP varia ao longo da existência de uma conexão, influenciando diretamente em alguns comportamentos observados no tráfego HTTP. Outros algoritmos e características importantes do TCP podem ser vistos em [31, 32, 33, 34].

Capítulo 3

Projeto do Modelo de Tráfego

NESSE capítulo é apresentado o modelo de tráfego HTTP baseado no conceito de agregação [35]. É apresentada uma descrição de suas principais características, como foi desenvolvido e avaliado.

3.1 Modelo de Tráfego

No capítulo anterior foram apresentados os principais problemas encontrados no desenvolvimento de modelos de tráfego *Web*, ao mesmo tempo em que foi destacada a importância de se ter tais modelos para a geração de tráfego em simulações. Como foi apresentado, o ponto de partida de um modelo é o seu objetivo, ou seja, qual aplicação ele vai descrever e qual seu enfoque ou quais características se espera que o modelo tenha. Portanto, é importante estabelecer um perfil que o modelo deve preencher para então desenvolvê-lo. Apesar de simples, ou até mesmo óbvia, essa metodologia pode minimizar o tempo de elaboração do modelo, à medida que focaliza o esforço de desenvolvimento e tenta evitar complexidades desnecessárias.

Nesse trabalho, foi estabelecido que o modelo deveria ser parametrizável de forma o permitir a simulação de diferentes condições de carga e que o número de parâmetros fosse reduzido. Para manter a simplicidade, o modelo poderia desconsiderar vários detalhes de clientes *Web* individuais, desde que o comportamento de um agregado de fontes fosse descrito apropriadamente. O principal comportamento a ser descrito pelo modelo seria a capacidade de representar uma carga controlada no sistema. A seguir são apresentados as motivações para essa escolha.

Os geradores de tráfego são às vezes chamados de geradores de carga (*workload generator*), pois geram uma carga de trabalho para a rede. Porém, o ajuste da carga desejada utiliza, em geral, métodos pouco eficientes. Um exemplo típico é o uso da carga média aproximada que um cliente ou um agregado representa para uma determinada configuração da rede. Para variar a carga, varia-se o número de clientes ou agregados. Porém, se a configuração da rede é alterada, é necessário recalcular a nova carga média representada pelo cliente ou agregado. Se uma configuração do cliente é também alterada, por exemplo o tamanho dos objetos *Web*, a carga imposta se modifica. Além disso, a carga média é medida em um longo intervalo de tempo, sendo que medições em intervalos menores podem apresentar valores que se desviam desse valor médio. No modelo proposto neste trabalho, o primeiro objetivo é permitir um ajuste simples da carga sob diferentes condições de largura de banda passante e o segundo é permitir que em diferentes escalas de tempo os valores da carga média sejam respeitados. O primeiro objetivo diz respeito ao interesse em realizar avaliações sob condições conhecidas, e facilmente configuráveis, de carga. O segundo se justifica na necessidade de se realizar avaliações nas quais realmente há a carga esperada em diferentes escalas de tempo ou valores próximos. O intuito é evitar conclusões incorretas utilizando o valor médio de uma longa escala de tempo que não corresponda ao comportamento em escalas de tempo menores.

Utilizando o conceito de carga ou fator de utilização de teoria de filas [36], tem-se:
$\rho = R/C$
onde,
R - taxa na qual o trabalho chega, e
C - taxa máxima (ou capacidade) na qual o sistema pode realizar o trabalho.

O trabalho que um novo consumidor [1] traz para o sistema é igual a quantidade

[1]Para não confundir com o **cliente** usado ao se referir ao *software Web*

de tempo de serviço que ele requisita. Ou seja, no caso de um único servidor (por exemplo um roteador em um enlace de gargalo), a carga pode ser descrita como:
$\rho = \lambda\overline{x}$
sendo que,
λ - taxa média de chegada de consumidores, e
$\overline{x}$ - tempo médio de serviço.
Para o tipo de uso pretendido nesse trabalho, tem-se:
$\overline{x} = \overline{L}/C$
Onde,
$\overline{L}$ - tamanho médio de uma transferência, e
C - taxa máxima (ou capacidade) na qual o elemento de rede consegue transmitir.
Logo,

$$\rho = \lambda\overline{L}/C \tag{3.1}$$

ρ é o principal parâmetro de ajuste do modelo e é usado para escolher diferentes condições de carga no enlace de gargalo. É importante observar que C é fixo para uma determinada configuração da rede, isto é, qualquer que seja o tamanho da transferência $\overline{L}$, ela sempre será transmitida a mesma taxa. Isso implica que além de descrever o percentual de C utilizado, ρ também representa o percentual de tempo que o sistema está ocupado dentro de uma janela de medição. Na seção seguinte, esse conceito será explorado em mais detalhes.

$\overline{L}$ é o tamanho médio de uma conexão *Web*. $\overline{L}$ pode descrever o tamanho de páginas ou objetos *Web*, para o caso de se desejar modelar HTTP/1.0 sem *keep-alive*; ou o tamanho de um agrupamento de páginas e objetos *Web* se o HTTP/1.0 com *keep-alive* ou HTTP/1.1 estiver sendo modelado. Na verdade, essa é uma aproximação permitida pelo caráter do modelo de agregado. $\overline{L}$ é expresso em *bytes* ao longo do texto. Como será apresentado posteriormente, apesar das diferenças numéricas dos dois tipos de modelagem de $\overline{L}$, as distribuições marginais e as suas conseqüências se mantém semelhantes. C é a capacidade de transmissão do enlace de gargalo, descrito em Mbytes/seg. Por fim, λ descreve a taxa de chegada de conexões e, de acordo com a proposta do modelo, deve variar de acordo com $\overline{L}$ de forma que o valor estabelecido para ρ seja atendido. Ou seja, o modelo pode ser descrito por:

$\lambda = \rho C / \overline{L}$

Uma vez que,

$T = 1/\lambda$

descreve o intervalo entre chegada de conexões, tem-se:

$$T = \overline{L}/\rho C \tag{3.2}$$

As distribuições que descrevem o parâmetro $\overline{L}$ tem sido amplamente estudadas [4, 7, 24, 25] e, apesar das diferenças dos resultados, há uma convergência nos trabalhos. A maior parte concorda que esse parâmetro é descrito apropriadamente por uma distribuição de cauda longa. A tabela 3.1 apresenta as distribuições obtidas por alguns estudos usados como referência neste trabalho. A informação de rótulo da tabela é usada para futuras citações dessas distribuições.

Tabela 3.1: Algumas distribuições que descrevem o parâmetro $\overline{L}$.

Distribuição	Configuração	Rótulo	Referência
Pareto	média - 4100 *shape* - 1.95	HTTP-1	[24]
Pareto	média - 4100 *shape* - 1.35	HTTP-2	[24]
Lognormal	média - 4827 desvio padrão - 41008	HTTP-3	[4]
híbrida: Pareto - 7%, Lognormal - 93%	média - 1463000 *shape* - 1.1	HTTP-4	[5]
	média - 27600 desvio padrão - 59714		
híbrida: Pareto - 12%, Lognormal - 88%	média - 10558 *shape* - 1.383	HTTP-5	[26]
	média - 7247 desvio padrão - 28765		

3.2 Estudo da Carga na Rede

A forma como se usa o termo "carga na rede" pode, às vezes, causar alguma confusão. Intuitivamente, é possível pensar em "carga na rede" como sendo a quantidade consumida de recursos do sistema. Por exemplo, se o sistema é um roteador e o recurso é banda passante disponível no enlace de saída, a carga corresponde a um consumo desse recurso. Ou seja, uma carga implica em um consumo da banda passante, que pode variar de 0 a 100% de toda a vazão do enlace de saída. Apesar de algumas situações permitirem essa abstração, o que acontece de fato é diferente.

A capacidade do enlace (ou enlaces) de saída de um roteador (ou outro equipamento de rede) é tipicamente fixa, ou seja, independente da taxa que os bits chegam a um roteador, eles sempre vão sair utilizando a taxa máxima (e única) do enlace de saída. Dessa forma, para que haja maior rigor na descrição do processo, o conceito de consumir um percentual da banda passante deve ser substituído pelo de consumir um percentual de tempo do roteador (ou enlace). Uma vez que a carga passa a ser medida como tempo de uso do sistema, é necessário introduzir a idéia de intervalo ou janela de medição. O intervalo de medição corresponde a uma quantidade de tempo durante a qual se verifica qual é a carga na rede. Inicialmente, o intervalo de medição pode ser arbitrário e variar dentro de uma ampla escala de tempo. No entanto, como já foi dito, muitas vezes é importante que a carga média esteja próxima do valor estabelecido em várias dessas escalas de tempo.

Dessa forma, para que ρ represente a carga efetiva no enlace, de acordo com a equação 3.1, é necessário que o sistema esteja ocupado durante o percentual correspondente de tempo de um determinado intervalo de medição, escolhido arbitrariamente. Por exemplo, para $\rho = 0,9$ (90%) e um intervalo de medição de 10 segundos, significa que o sistema deveria estar ocupado por 90% desse, ou seja, 9 segundos. Surge então duas considerações importantes sobre o modelo.

Primeira, foi estabelecido que em várias escalas de tempo, o valor da carga deveria se aproximar da média desejada. Isso seria afetado pelo tamanho das transferências porque, intuitivamente, conexões de curta duração poderiam respeitar o valor configurado em curtos intervalos de medição, enquanto transferências de longa duração exigiriam extensos intervalos. Além disso, o HTTP utiliza o TCP como protocolo de transporte, que por sua vez faz com que as transferências cheguem a uma taxa variável no enlace de gargalo, o que criaria "distorções" nas seqüências de uso do sistema. Dessa forma, o intervalo de medição seria novamente afetado.

Segunda, o protocolo TCP é confiável e retransmite pacotes perdidos devido, tipicamente, ao transbordo de *buffers*. Se mais de um exemplar de determinado pacote passasse pelo ponto da rede onde a carga esta sendo medida, a carga seria superior a ρ, pois o consumo de tempo do sistema seria superior ao especificado. Por outro lado, como o modelo é projetado para enlaces de gargalo, as medições são realizadas nesse ponto, e portanto, as perdas deveriam ocorrer ao chegar no *buffer* desse enlace e duplicatas não seriam contabilizadas.

Para avaliar as considerações acima foram realizadas simulações que verificam a relação entre ρ e a carga média efetiva medida na rede em diferentes escalas de tempo. Nessas simulações foram utilizadas fontes de tráfego com transferências de tamanho fixo iguais a 1, 5, 50, 500 e 5000 Kbytes, e também com as distribuições apresentadas na tabela 3.1.

A topologia escolhida para as simulações é uma variação da tradicional *dumbbell* e é apresentada na figura 3.1. Vários trabalhos utilizam essa topologia, que apesar da simplicidade tem se mostrado apropriada para uma ampla variedade de avaliações. O tamanho do *buffer* do enlace segue o recomendado por [37, 38], ou seja, $2 * Bw * RTT$, onde $Bw = C$ e RTT é o atraso de propagação de ida e volta mais longo da rede. Esse tamanho de *buffer* permite que o TCP consiga consumir toda banda passante e manter a memória da rede ($Bw * RTT$) cheia. Na verdade, há controvérsias sobre o valor ideal, mas grande parte dos trabalhos tem utilizado a configuração anterior com sucesso. A implementação do TCP utilizada é Reno, por ser uma das versões mais utilizadas na Internet. Na topologia utilizada, a capacidade do enlace de acesso (M) determina a taxa de pico (ou taxa máxima) na qual rajadas emitidas pelas fontes chegam aos nós de agregação s_i. Quando a capacidade deste enlace de acesso é maior que a capacidade do enlace de gargalo, a taxa de pico pode ser considerada como infinita. Caso contrário, ela limita a banda passante obtida. A fila do roteador é do tipo FIFO e o mecanismo de "gerenciamento" da fila é o *Drop tail*, ou seja, a configuração tradicionalmente usada na Internet.

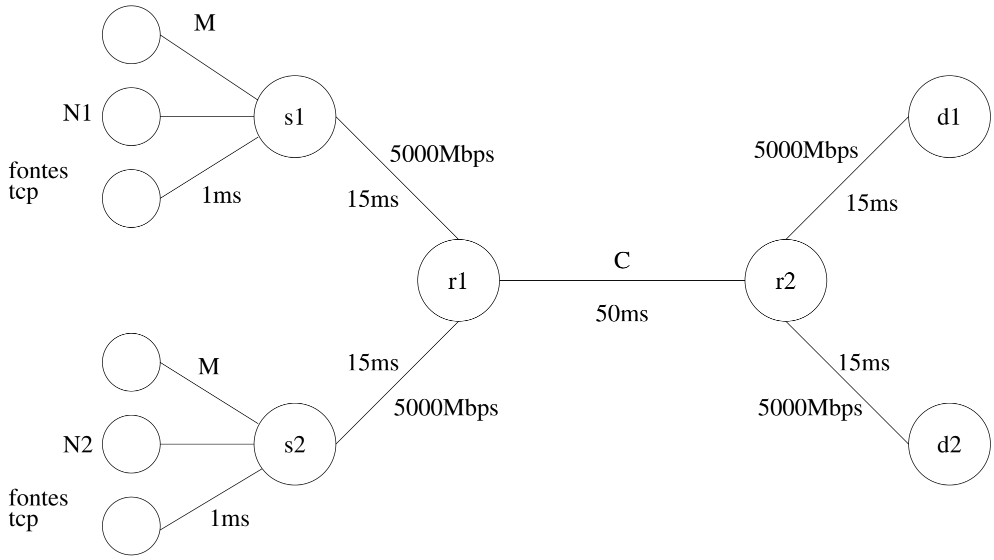


Figura 3.1: Topologia de rede usada nas simulações.

As figuras 3.2 e 3.4 apresentam os resultados obtidos quando ρ é calculado com base em um enlace de capacidade C e o enlace é configurado com essa capacidade. As figuras 3.3 e 3.5 exibem os resultados quando ρ é calculado, novamente, com base em um enlace de capacidade C, porém o enlace tem uma capacidade efetiva de $10C$. O intuito desse último cenário é avaliar se o gerador obedece a configuração de ρ para valores maiores que 100%. Nas figuras 3.2 a 3.5 a carga indicada corresponde à média medida no intervalo de 50 segundos até o fim da simulação, que ocorre em 500 segundos. Os primeiros 50 segundos são descartados para eliminar o transiente. Para as figuras 3.2 a 3.5, os experimentos utilizaram transferências de tamanho fixo com os seguintes valores: 1, 5, 50, 500 e 5000 Kbytes.

As figuras 3.3 e 3.5 mostram valores de ρ que excederiam a capacidade do enlace, porém efetivamente este era 10 vezes maior. Essas simulações foram importantes para verificar se a relação entre ρ e carga se manteria conforme esperado. Por outro lado, quando a capacidade é efetivamente a mesma utilizada no cálculo de ρ, as cargas, por definição, ficam limitadas a 100%.

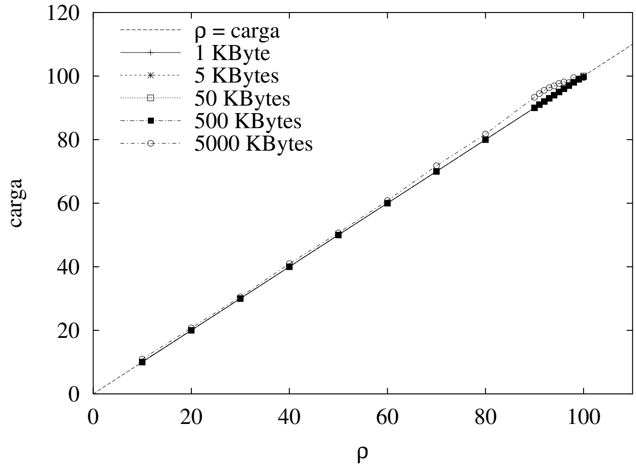


Figura 3.2: Relação entre carga e ρ para enlace de capacidade C.

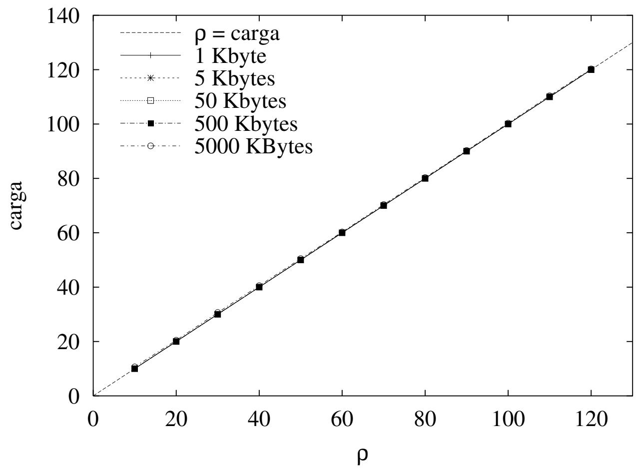


Figura 3.3: Relação entre carga e ρ para enlace de capacidade 10C.

As figuras 3.2 e 3.3 mostram que para as transferências de 1, 5, 50 e 500 Kbytes há uma correspondência exata entre ρ e carga. Porém, para as transferências de 5000 Kbytes no enlace de capacidade C, a carga é superior a ρ a partir de 70%. O mesmo não ocorrendo no enlace de capacidade $10C$. Experimentos mostraram que isso acontece devido ao comportamento de rajada do TCP combinado com seu controle de taxa. Como as transferências são longas o suficiente para manter o sistema ocupado por um extenso período de tempo, o TCP tem oportunidade de aumentar sua janela de transmissão para além da capacidade da rede, provocando o transbordo do *buffer*. As perdas geram retransmissões e diminuem a taxa efetiva (*goodput*) das

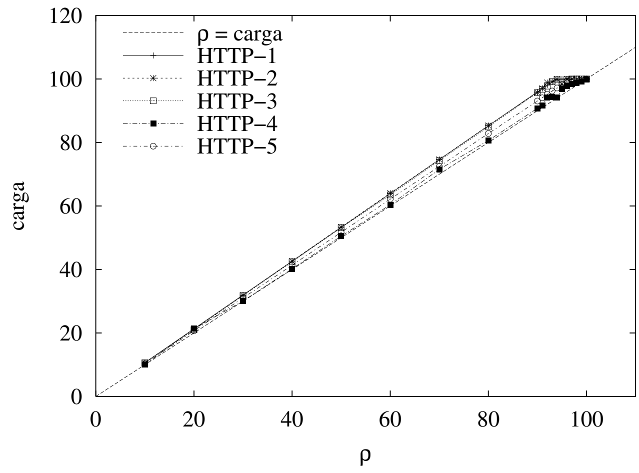


Figura 3.4: Relação entre carga e ρ para enlace de capacidade C.

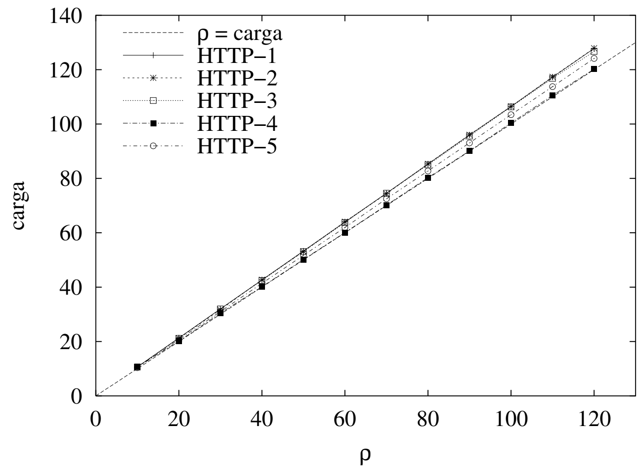


Figura 3.5: Relação entre carga e ρ para enlace de capacidade 10C.

conexões. Isso faz com que as transferências se espalhem mais no tempo e aumentem o número de conexões simultâneas. Dessa forma, o sistema fica com uma "dívida" de tempo ocioso que não consegue liquidar até o fim da simulação, o qual ocorre de forma abrupta após um tempo preestabelecido. Para que o comportamento seja observado é necessário que ocorram rajadas e que haja seqüências de conexões de longa duração. Vale lembrar que o conceito de longa duração é relativo e diz respeito à razão entre o tamanho da transferência e a capacidade do enlace. De certa forma, isso quer dizer que o modelo se torna menos sensível a rajadas à medida que a capacidade do enlace aumenta e as conexões se mantém do mesmo tamanho.

A tabela 3.2 facilita a compreensão do fenômeno descrito anteriormente. Para obtenção dos valores apresentados, foram realizadas simulações nas quais o critério de encerramento é a finalização de um determinado número de transferências. O número mínimo de 50 transferências foi escolhido para garantir que pelo menos 500 segundos de simulação fossem executados. O ρ escolhido foi a partir de 70%, pois a partir desse ponto é possível observar uma certa discrepância entre ρ e carga, de acordo com a figura 3.2. Além disso, foram utilizadas duas configurações de *buffer*, uma idêntica a usada na obtenção dos resultados da figura 3.2 e outra que possui um *buffer* 10 vezes maior. A última configuração visa oferecer espaço suficiente para absorção das longas rajadas. Os valores apresentados na tabela 3.2 representam a média entre 10 simulações executadas para cada configuração de número de transferências.

A partir da tabela 3.2 é possível observar que a quantidade de bytes transferidos durante os primeiros 500 segundos de simulação (menos o transiente) é sempre superior para o *buffer* B, assim como a de pacotes perdidos. Isso ocorre porque, na primeira configuração, a taxa de transmissão efetiva diminui, prolongando o tempo necessário para realizar as transferências. Dessa forma, diminui o tempo de ociosidade do sistema, elevando a carga média na rede. Essa informação é confirmada pelo tempo em que não há nenhuma transferência em andamento, o qual é significativamente superior para a configuração com *buffer* $10B$ também mostrado na tabela 3.2.

A figura 3.3 mostra que se rajadas são absorvidas, através do aumento da capacidade do enlace, o modelo não sofre distorções. Como foi mostrado anteriormente, outra forma de verificar que as rajadas contribuem para o fenômeno de $carga > \rho$ é através do aumento do *buffer* do enlace de gargalo. A figura 3.6 mostra a relação entre ρ e carga para um enlace de capacidade efetiva C, porém com um buffer de capacidade 10 vezes maior que o original. Dessa forma, as rajadas são absorvidas pelo *buffer* e a relação entre ρ e carga é mantida conforme esperado.

As figuras 3.4 e 3.5 apresentam, persistentemente, cargas superiores às configurações de ρ. Esse comportamento se deve a uma "perda por fragmentação" ocasionada

Tabela 3.2: Comportamento das conexões de 5M ao variar o *buffer*.

	bytes transmitidos (Kbytes)		Pacotes perdidos (Pacotes)		Tempo sem transferências (Segundos)		Maior # de transferências simultâneas	
ρ	B	10B	B	10B	B	10B	B	10B
70	405164	395357	29679	0	36,01	100,01	2	2
75	429480	421710	21353	0	29,25	70,49	2	2
80	461326	450000	22854	0	0	51,30	5	2
85	499669	479357	21790	0	0	23,41	5	2
90	525367	506267	22256	0	0	1,31	6	2

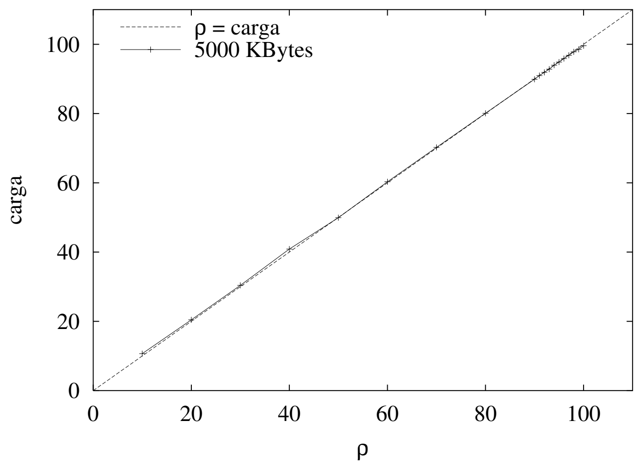


Figura 3.6: Relação entre carga e ρ para enlace de capacidade 10B.

por uma característica do simulador NS. No NS, cada fonte de tráfego, ao ser criada, recebe uma configuração do tamanho do pacote [2] que usará em todas as transferências que realizar na simulação corrente. Se o tamanho de uma transferência é um múltiplo do tamanho do pacote, não há problemas. Caso contrário, o último pacote de cada transferência gerará um desperdício de banda passante, pois serão transmitidos mais bits do que seriam necessários para completar a transferência em

[2] No NS a sobrecarga dos cabeçalhos é, tipicamente, ignorada e o MSS é fixo e, portanto, equivalente ao pacote IP

questão. Por exemplo, se o tamanho de pacote escolhido para uma fonte for 1000 bytes e uma transferência tiver o tamanho de 1200 bytes, serão transmitidos 2000 bytes ou 2 pacotes de 1000 bytes. Em medições recentes [39], foi observado que um tamanho comum de pacote de dados IP na Internet é 552 ou 576 bytes. O tamanho de pacote escolhido para as simulações foi de 500 bytes, por ser próximo ao pacote típico da Internet e por ser um múltiplo do tamanho das transferências de tamanho fixo utilizadas. A figura 3.7 mostra o comportamento da carga ao variar o tamanho das transferências de 100 a 500 bytes. É possível observar que quanto maior é o desperdício, maior é a carga observada.

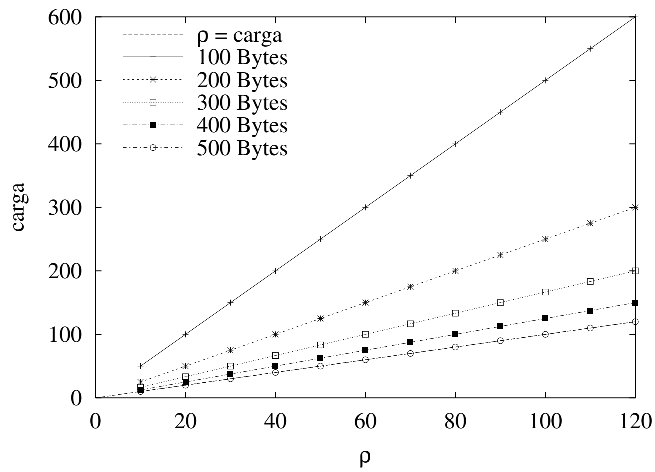
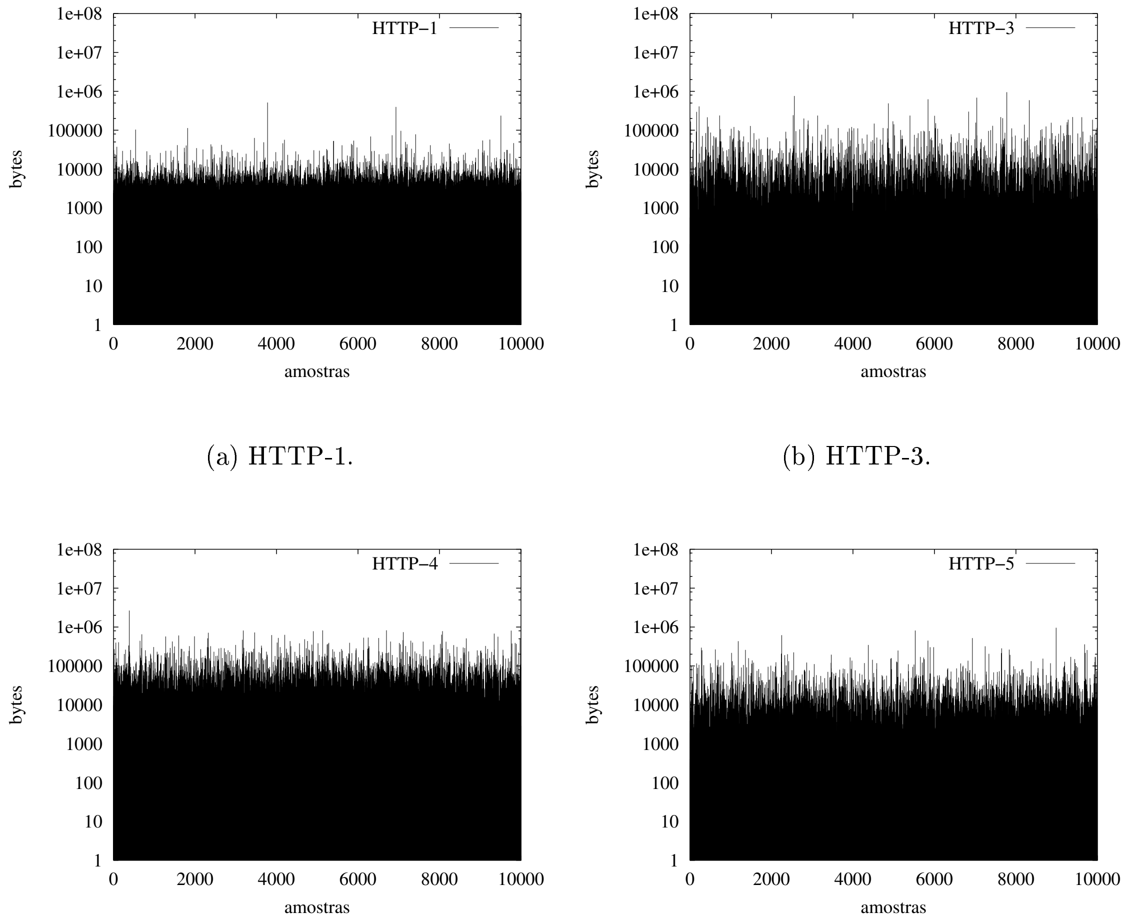


Figura 3.7: Proporcionalidade da carga em relação ao tamanho das transferências.

3.2.1 Tamanho das Transferências

Como foi apresentado, várias seqüências de longas transferências podem gerar alguma distorção no modelo. Ainda que as discrepâncias encontradas não sejam significativas, é interessante observar se seria comum a ocorrência desse comportamento no tráfego HTTP. Para tanto, foram avaliadas as distribuições do tamanho de transferências de alguns trabalhos amplamente usados como referências. As figuras 3.8(a) a 3.8(d) mostram exemplos de seqüências típicas dos valores gerados utilizando as distribuições de tamanho de transferência encontradas em [24], [5], [26] e [4], respectivamente.

Como pode ser observado nas figuras, a ocorrência de transferências de longa duração não é muito freqüente e não há tendência a ocorrerem em seqüência. De fato, a maior parte das transferências tem menos que 50 Kbytes. Isso é confirmado pela tabela 3.3 que mostra o percentil de alguns tamanhos típicos de transferências. Nessa tabela estão descritos os resultados de 100 seqüências de cada uma das distribuições escolhidas, sendo que cada seqüência tem cem mil amostras. Como pode ser visto, é relativamente baixo o número de transferências de longa duração.



(a) HTTP-1. (b) HTTP-3.

(c) HTTP-4. (d) HTTP-5.

Figura 3.8: Exemplos de distribuições avaliadas.

Tabela 3.3: Percentil das distribuições.

	HTTP-1	HTTP-2	HTTP-3	HTTP-4	HTTP-5
< 1K	0.00000%	0.00000%	60.84710%	3.15523%	36.64540%
< 5K	83.30260%	87.63150%	85.38030%	26.20360%	73.18730%
< 50K	99.81060%	99.45360%	98.48300%	86.63730%	97.66280%
< 500K	99.99800%	99.97560%	99.94760%	99.78430%	99.96020%
< 5000K	100.00000%	99.99900%	99.99940%	99.99980%	99.99990%

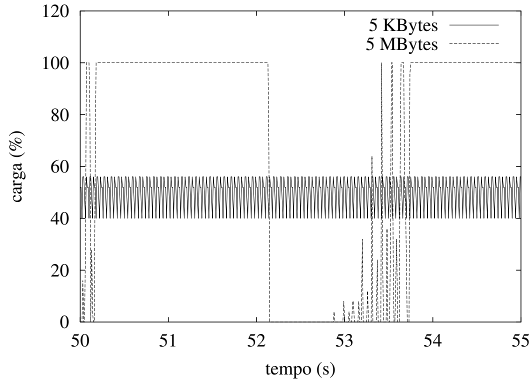
3.2.2 Relação entre ρ e Carga

Para avaliar a partir de qual escala de tempo a relação $carga = \rho$ se torna válida, foram realizadas simulações nas quais a carga média é medida em diferentes intervalos de tempo. As figuras 3.9(a) a 3.9(d) mostram alguns exemplos da variação da carga média ao longo do tempo a intervalos de 10 milisegundos, com $\rho = 50\%$. Como pode ser observado, nessa escala de tempo, a carga média apresenta uma grande variação.

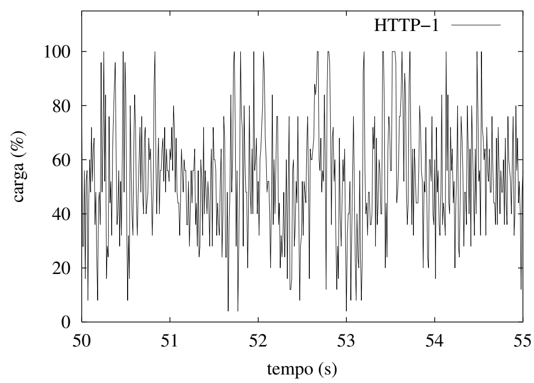
As figuras 3.10(a) e 3.10(b) mostram a variação da carga média ao longo do tempo em intervalos de 1 segundo. Ainda há uma variação significativa, porém já é possível observar uma tendência na média de seguir o valor configurado para ρ. As figuras 3.10(c) a 3.10(f) apresentam a carga média medida em intervalos de tempo de 10 e 50 segundos. É possível observar que a partir de 10 segundos a carga média se aproxima de ρ. Ou seja, os resultados mostram que o modelo cumpre o seu segundo objetivo que é ter, deterministicamente, uma carga próxima a ρ a partir de uma escala de tempo em torno de 10 segundos, nas configurações avaliadas.

3.3 Conexões Simultâneas

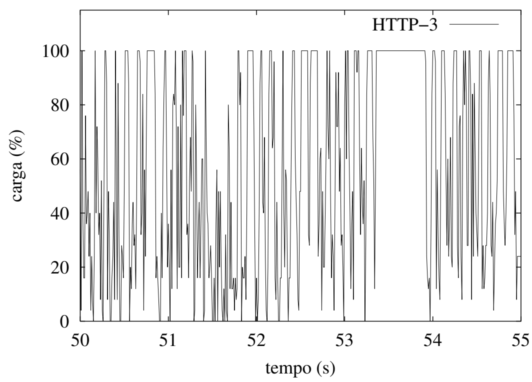
Nessa seção é apresentado um estudo sobre a simultaneidade de conexões, como elas ocorrem quando se utiliza o modelo HTTP de agregado e a influência dos



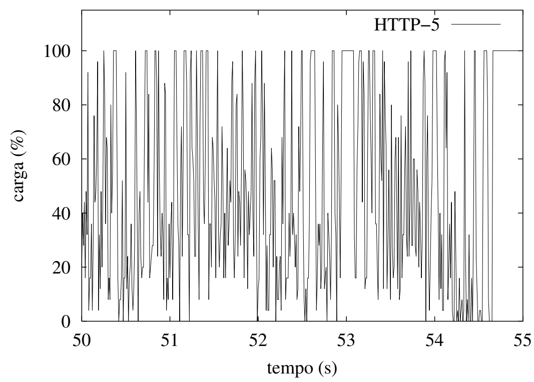
(a) Transferências de tamanho fixo.



(b) HTTP-1.



(c) HTTP-3.



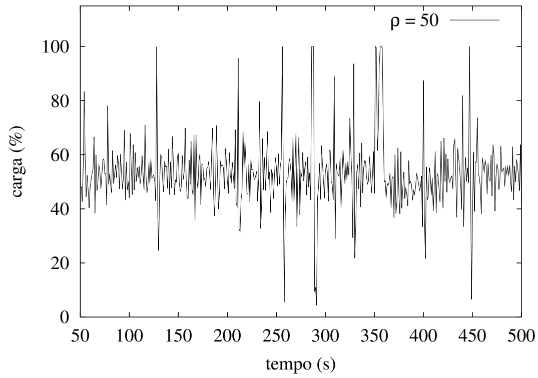
(d) HTTP-5.

Figura 3.9: Variação da carga média em intervalos de 10 milisegundos.

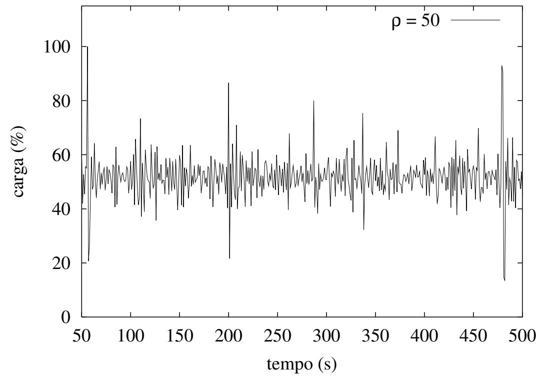
parâmetros do modelo. Como será mostrado, esse modelo possui a capacidade de controlar indiretamente o número de conexões simultâneas através do controle de ρ.

Inicialmente, o modelo proposto se baseou no HTTP/1.0, porém se algumas simplificações forem feitas é possível descrever o HTTP/1.1, conforme citado anteriormente. Nesse contexto, uma conexão será tratada como uma transferência, que poderia representar um única página/objeto *Web* ou várias.

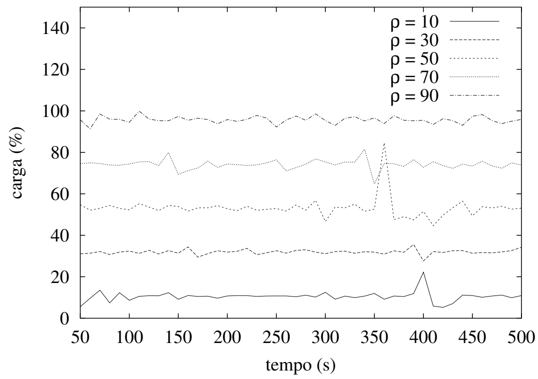
Caso o protocolo de transporte utilizado fosse capaz de utilizar toda a capacidade que lhe é atribuído de forma imediata, o número de conexões simultâneas seria sempre menor ou igual a dois para cargas menores que 100%. Ou seja, no máximo existiriam ao mesmo tempo uma conexão vindo s_1 e outra de s_2. No entanto, o



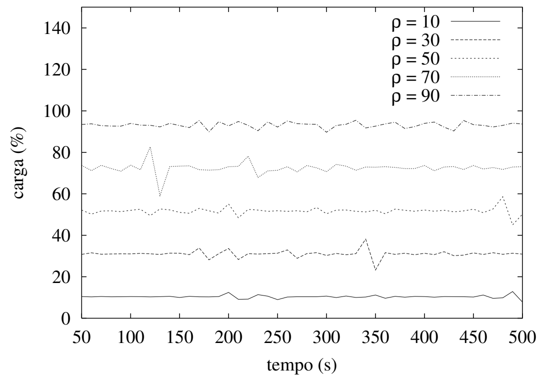
(a) HTTP-3.



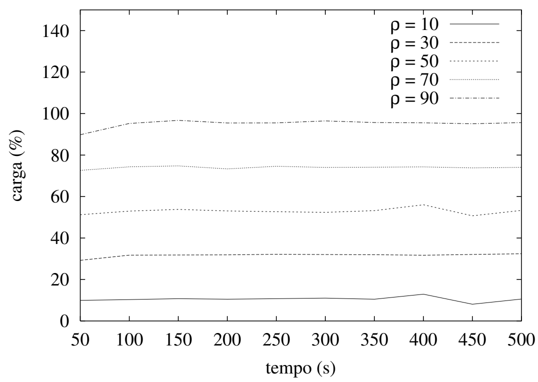
(b) HTTP-5.



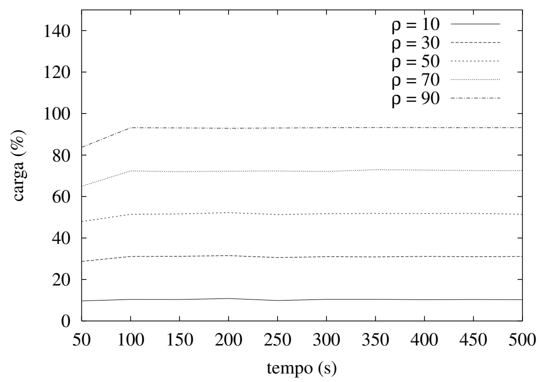
(c) HTTP-3.



(d) HTTP-5.



(e) HTTP-3.

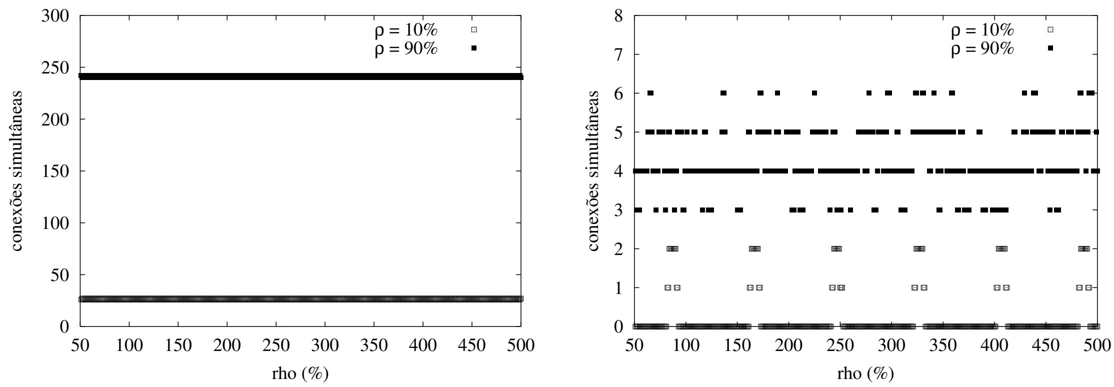


(f) HTTP-5.

Figura 3.10: Variação da carga média em intervalos de 1, 10 e 50 segundos.

gerador de tráfego HTTP do modelo utiliza o TCP como protocolo de transporte, que por sua vez não é capaz de usar de forma imediata toda capacidade de um enlace, como pode ser visto detalhadamente em [31, 34].

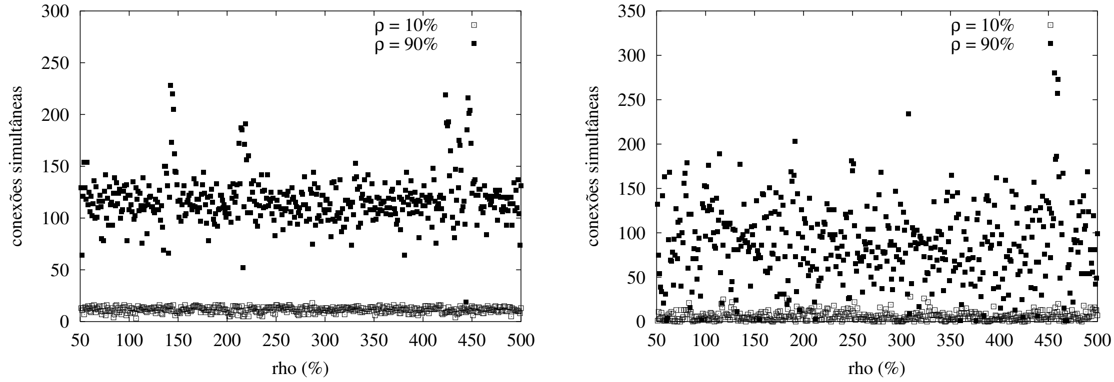
Assim, o protocolo TCP contribui de forma especial no número de conexões simultâneas. Primeiro, devido ao algoritmo *slow start*, as transferências de curta duração têm uma baixa vazão [13] e, dada sua breve existência, apresentam ampla sobreposição. Esse fato pode ser observado na figura 3.11(a) e também nas figuras 3.12(a) e 3.12(b), pois há predominância de conexões de curta duração. Por outro lado, o número de transferências simultâneas para conexões de longa duração é significativamente menor, como seria esperado, uma vez que a vazão de conexões de longa duração tende a ser alta. Isso é ilustrado na figura 3.11(b).



(a) Transferências de 1K. (b) Transferências de 5M.

Figura 3.11: Número de conexões simultâneas para transferências de tamanho fixo.

Segundo, o TCP possui um controle de congestionamento que regula a taxa de transmissão de acordo com a ocorrência de notificações de congestionamento (perdas ou pacotes com ECN configurado). Essa regulagem faz que com que o número de conexões simultâneas aumente à medida que a carga na rede cresce. Como um aumento na carga implica em um aumento na taxa de chegada de novos clientes, cresce também a probabilidade de perdas, retransmissões e duração das conexões. Tais conexões, incrementam o número de transferências simultâneas. A figura 3.13 ilustra essa situação. Como pode ser observado, à medida que a carga se



(a) HTTP-1.

(b) HTTP-3.

Figura 3.12: Número de conexões simultâneas para transferências HTTP.

aproxima de 100%, o número de conexões simultâneas aumenta proeminentemente. Esse fenômeno é mais destacado nas transferências HTTP, pois elas combinam uma grande quantidade de conexões curtas com algumas longas.

A partir da figura 3.13 mostra também que ρ funciona como um ajuste indireto no número de conexões simultâneas. Esse recurso é útil em situações nas quais esse número precisa ser ajustado. Um exemplo é a avaliação de mecanismos de escalonamento que lidam com grande quantidade de fluxos simultâneos. Através do ajuste apropriado de ρ é possível obter uma ampla variedade do número de fluxos simultâneos, desde algumas unidades até milhares.

3.4 Estudo da Auto-Similaridade

Alguns trabalhos mostram que o tráfego HTTP coletado e avaliado em vários pontos da Internet apresenta auto-similaridade [5, 6, 7, 25, 26]. Eles recomendam ainda que modelos de tráfego HTTP levem em consideração essa característica, pois o impacto no desempenho da rede é significativo. De forma genérica, auto-similaridade corresponde ao fenômeno no qual uma determinada propriedade de um objeto é preservada independentemente da escala do espaço ou do tempo. Em tráfego de rede, a manifestação mais estudada desse fenômeno é a ocorrência de rajadas em

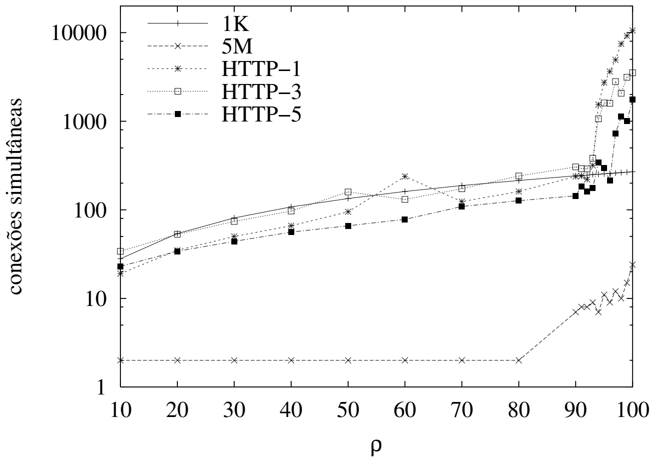


Figura 3.13: Número de conexões simultâneas em relação à carga.

várias escalas de tempo. Vários trabalhos abordam a existência e conseqüências da auto-similaridade em redes computadores [5, 24, 25, 40, 41].

Em [24, 41] é apresentado mais detalhes sobre como a auto-similaridade do tráfego HTTP pode afetar o desempenho da rede. De acordo com esses trabalhos, o aumento da auto-similaridade implica no aumento da taxa de perda de pacotes, elevação da taxa de retransmissão do TCP, e sobretudo, aumento significativo no atraso nas filas. Por outro lado, existe outra abordagem [42] que discute até que ponto a auto-similaridade afeta a rede, uma vez que os *buffers* são limitados e, portanto, um limite máximo conhecido para o atraso dos pacotes. Além disso, o *buffer* limitado tenderia a restringir a auto-correlação entre as transferências e, portanto, limitar o comportamento de rajada e suas conseqüências.

Em [24, 25, 42, 43, 44] são apresentadas algumas justificativas para o comportamento auto-similar do tráfego HTTP. A maior parte dos autores defende a idéia de que distribuições de cauda longa para o tamanho das transferências e agregação dos fluxos são os principais responsáveis pelo fenômeno. Outros acreditam que o papel do protocolo de transporte é fundamental. Ainda outros estudam a possibilidade do fator humano ao utilizar a *Web* ser um fator importante na auto-similaridade observada no tráfego. [45] mostra como a auto-similaridade pode se propagar ao longo de redes interconectadas. No trabalho, os autores mostram que é suficiente o compartilhamento de enlaces para que um tráfego auto-similar espalhe essa característica para outros tráfegos.

No entanto, em [8] os autores questionam se a auto-similaridade se apresentaria em enlaces da espinha dorsal da Internet, uma vez que o nível de agregação é muito alto e a variância no processo de chegada de pacotes é baixo. Um trabalho recente [46] confirma as suspeitas do anterior ao avaliar uma grande quantidade de dados coletada a partir de alguns pontos da espinha dorsal da Internet. No entanto, esse trabalho não invalida os demais, pois em situações onde nível de agregação não seja tão elevado, os resultados a respeito da auto-similaridade continuam válidos.

A auto-similaridade e suas diferentes definições estão formalmente descritos em alguns trabalhos [40, 47, 48, 49]. Nesse trabalho, a auto-similaridade de interesse é a assintótica de segunda ordem, tipicamente encontrada no tráfego HTTP. O parâmetro Hurst (H) é a principal forma de avaliação da auto-similaridade e esta definido no seguinte intervalo: $0,5 < H < 1$. Quanto mais próximo de 1, maior é a auto-similaridade. A medição do parâmetro Hurst foi feita utilizando um estimador baseado em *wavelets*, apresentado em [50].

Nas medições de auto-similaridade realizadas, cada amostra consiste em uma seqüência de tempo de 3600 segundos e nos bits transmitidos nesse intervalo. O processo é discretizado em janelas de 10 milisegundos e, posteriormente, é gerado o processo acumulativo do trabalho. Ou seja, é gerada uma seqüência acumulativa dos bits transmitidos, que cresce assintoticamente. Esse processo é fornecido como entrada para o estimador baseado em *wavelets* [51], o qual é capaz de fornecer o parâmetro de Hurst e as escalas de tempo em que ocorrem as rajadas.

Foi observado que o modelo HTTP de agregado é capaz de reproduzir diferentes níveis de auto-similaridade. Por exemplo, em uma das medições realizadas foi observado que o processo apresentava "forte" auto-similaridade com $H = 0.889$ e rajadas que variavam de alguns segundos até centenas de segundos, ou seja, duas ordens de magnitude. Além desse exemplo, outros valores de H foram obtidos. No entanto, o modelo HTTP de agregado não apresenta uma forma simples e precisa de controlar o nível de auto-similaridade gerado. Há indicações de quais combinações de carga e distribuições são capazes de produzir auto-similaridade no tráfego gerado, porém o modelo não é capaz de fornecer a priori qual o valor de H e as escalas de tempo que serão produzidas ao configurar os parâmetros.

3.5 Implementação do Modelo

Os resultados obtidos nesse trabalho se basearam nos dados coletados a partir do simulador NS-2 [9]. Para tanto, o modelo de tráfego foi implementado no simulador em conjunto com recursos para medição e avaliação. Nessa seção é feita uma breve descrição da implementação, enquanto o código completo de uma versão reduzida do gerador está disponível no apêndice A.

O simulador NS começou como uma variante do simulador de rede REAL e, posteriormente, foi apoiado pela DARPA através do projeto VINT (*Virtual InterNet Testbed*), do qual participaram LBL, Xerox PARC, UCB e USC/ISI. Atualmente, o NS-2 continua sendo apoiado pela DARPA através do projeto SAMAN (*Simulation Augmented by Measurement and Analysis for Networks*) e pela NSF através do projeto CONSER (*Collaborative Simulation for Education and Research*). O código fonte do simulador é aberto e livre para modificações e redistribuições. Isso tem atraído e incentivado pesquisadores do mundo inteiro a utilizar e contribuir.

O NS-2 é um simulador de rede do tipo eventos discretos, ou seja, tudo que ocorre em uma simulação tem origem em eventos que são agendados em um escalonador. No NS já estão implementados grande parte dos recursos utilizados em redes, sobre o que se refere à Internet. Isto inclui protocolos, mecanismos, elementos de rede, arquiteturas etc. Além disso, o NS permite a criação ou modificação de uma ampla variedade de recursos. O simulador foi implementado utilizando-se duas linguagens de programação: C++ e OTcl. Em C++ estão escritos os trechos do programa que exigem melhor desempenho, enquanto em OTcl estão escritas as demais partes e também a interface, ou seja, onde se cria os *scripts* para a simulação propriamente. OTcl é uma linguagem orientada objetos que é executada de forma interpretada, tornando-a menos eficiente computacionalmente que linguagens compiladas, como

C++. Por outro lado, OTcl possui recursos poderosos e um bom nível de abstração, o que a torna útil para uma ampla variedade de usos.

O gerador de tráfego foi implementado utilizando-se a linguagem OTcl. Foi criada uma classe chamada **TrafficGen** que é responsável por gerar o tráfego entre dois nós quaisquer. Os dois nós podem pertencer a uma topologia qualquer, sendo importante apenas saber o valor do enlace de menor capacidade (gargalo) entre os nós. De fato, qualquer valor diferente do enlace de gargalo pode ser utilizado, sendo tal configuração para manter a relação entre ρ e carga de acordo com o foi apresentado. No entanto, há situações em pode se desejar outro comportamento.

Uma vez configurado e instanciado um objeto **TrafficGen**, é suficiente executar o método **start** para ter o gerador de tráfego funcionando. Informações úteis como vazão média e número médio de conexões podem ser obtidas através do método **plot_results** do objeto **post** que é instanciado automaticamente pelo gerador a partir da classe **PostProcess/gnuplot**.

3.5.1 Portando o Código para o Linux

Uma versão baseada em *sockets* para Linux está sendo implementada e deve estar disponível em breve. Alguns testes preliminares foram realizados para verificar requisitos básicos do modelo como a capacidade de utilizar distribuições para descrever os tamanhos das transferências e simultaneidade das conexões. Mais avaliações serão realizadas para verificar se a implementação apresenta resultados semelhantes aos de simulação. Assim que os testes e correções apropriadas forem realizadas, essa versão do gerador de tráfego estará disponível no seguinte URL: **http://www.gta.ufrj.br/~kleber/tfg**.

De fato, pretende-se realizar vários testes e avaliações com o gerador de tráfego de tráfego dentro da infra-estrutura da RNP2. Dentro desse contexto, espera-se contar com recursos da rede para estudar o modelo desenvolvido, realizar estudos de casos semelhantes aos apresentados no capítulo 4 e desenvolver novas avaliações que contribuam nas pesquisas que são realizadas dentro da RNP2.

Capítulo 4

Aplicações do Modelo de Tráfego

PARA motivar o uso do modelo de agregado proposto, nesse capítulo são sugeridos e implementados alguns estudos de caso de utilização do mesmo, assim como uma detalhada avaliação e estudo do descarte seletivo de pacotes e das políticas de *push-out* usando-se o modelo HTTP de agregado [52, 53].

4.1 Onde Utilizar o Modelo de Agregado

Uma vez que o modelo de agregado foi desenvolvido para produzir uma carga equivalente a uma agregação de clientes HTTP, ele é adequado para determinados tipos de uso. Um dos usos mais proeminentes é como entrada para enlaces de gargalo, caracterizados tipicamente por roteadores ou *switches*. Ou seja, o modelo é útil para avaliar mecanismos, políticas ou disciplinas que sejam implementadas nesse tipo de equipamento. Alguns exemplos são:

- gerenciamento de fila, como RED, BLUE, REM etc.;
- escalonamento, como GPS, WFQ, WRR etc.;
- marcadores, condicionadores de tráfego etc.

Além disso, o modelo oferece uma forma simples de gerar cargas controladas com diferentes marcações, pertencentes a diferentes classes de serviço. Esse recurso é útil em arquiteturas de provisão de qualidade de serviço, como o DiffServ.

Como já foi mostrado, o modelo de tráfego permite um controle simples da carga, do tamanho das transferências e do número de conexões simultâneas. Isso torna o modelo útil para avaliações de algoritmos e mecanismos de controle de congestionamento. Além disso, as facilidades oferecidas pelo modelo o tornam um prático gerador de tráfego de fundo (*background traffic*). Esse é um recurso desejável quando se pretende estudar o comportamento de algum tipo de tráfego como voz ou vídeo sob a presença de tráfego HTTP.

Nas seções 4.2 e 4.3 são apresentados estudos de caso da utilização do modelo de tráfego na avaliação do controle de congestionamento do TCP e de mecanismos de fila, respectivamente. Os estudos foram elaborados com base em outros trabalhos, porém utilizando o modelo de HTTP de agregado. Esse é um ponto importante, sobretudo nas avaliações com mecanismos de fila, que são avaliados na maioria das vezes utilizando-se fontes TCP de longa duração. Os estudos anteriores são utilizados como referência para os resultados esperados e como parâmetros de comparação. Antes das avaliações é apresentada uma descrição breve do mecanismo ou fenômeno sendo avaliado, e então as simulações e resultados são apresentados e discutidos.

Na seção 4.4 é apresentado um estudo de avaliação de desempenho de mecanismos de descarte seletivo de pacotes que utilizam técnicas de *push-out* para a diferenciação de serviços na Internet. O objetivo desse estudo é avaliar o grau de diferenciação e a eficiência dos mecanismos de descarte seletivo no aproveitamento da banda passante disponível em diferentes condições de carga. São apresentados novos resultados sobre o uso de mecanismos de descarte seletivo do tipo *loss conserving* para diferenciação entre classes de serviço. Além disso, é avaliada a perspectiva de se associar políticas de remoção com mecanismos de gerenciamento de *buffer* de descarte prévio (tipo RED).

A topologia utilizada nos estudos é mostrada na figura 4.1. Na verdade, a figura é idêntica à utilizada no capítulo anterior e foi repetida para facilitar a consulta a mesma. Como anteriormente, o tamanho do *buffer* do enlace segue o recomendado por [37, 38], a implementação do TCP utilizado é Reno e a fila do roteador é do tipo FIFO. É importante observar que os pacotes que saem de s_1 e s_2 pertencem a duas classes de tráfego diferentes, ou seja, recebem uma marcação diferenciada. E, de fato, $\rho = \rho_1 + \rho_2$. Nos dois primeiros estudos de casos não há mecanismos que tratem de forma diferenciada os pacotes, logo não há influência da marcação nas avaliações. O recurso de marcação foi mantido para se ter um único ambiente de simulação para todos os experimentos, inclusive nos que há tratamento diferenciado de pacotes como ocorre no terceiro estudo de caso.

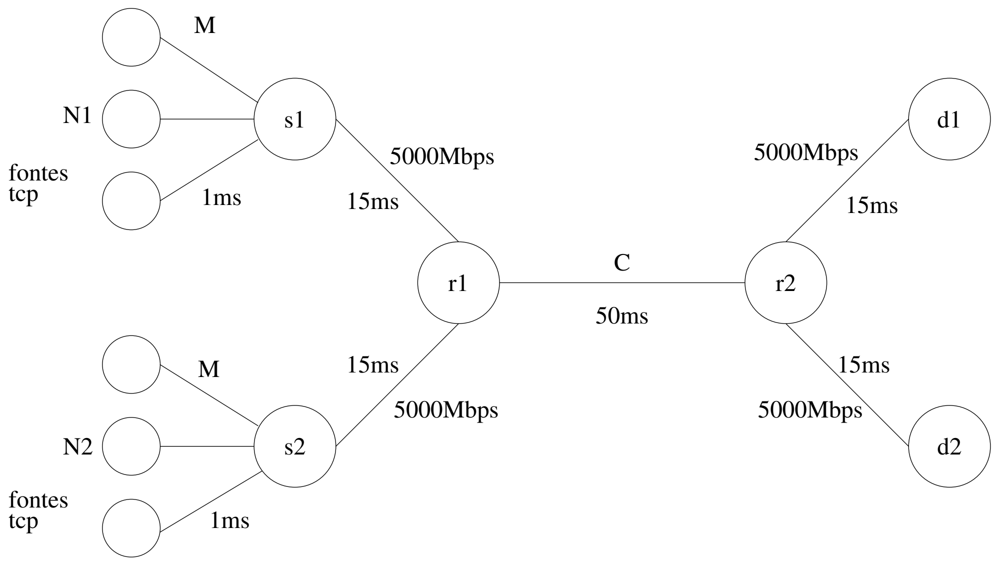


Figura 4.1: Topologia de rede utilizada nos estudos de caso.

Devido à característica aleatória de alguns componentes da simulação, principalmente do modelo de tráfego utilizado, houve uma preocupação em observar que os resultados obtidos tivessem um intervalo de confiança associado. Dessa forma, os valores apresentados nos gráficos possuem um intervalo de confiança de 95% e um erro de mais ou menos ±10% do valor médio. Para garantir o cumprimento dessas restrições foi utilizada uma técnica conhecida como replicações independentes [54]. Essa técnica consiste em realizar sucessivas simulações até que o intervalo de confiança desejado seja atingido. Pelo menos 5 simulações são executadas em cada avaliação, mesmo que o intervalo de confiança já tenha sido alcançado.

4.2 Influência do Controle de Congestionamento na Vazão

Alguns trabalhos [13, 14] têm estudado a diferença do comportamento das conexões TCP de curta e longa duração. A principal motivação desses trabalhos tem sido o número significativamente maior de conexões TCP de curta duração na Internet e sua relação com o controle de congestionamento do TCP. Como foi visto na seção anterior, no tráfego HTTP há uma predominância de páginas e objetos de "pequeno" tamanho.

Inicialmente se acreditava que as conexões TCP de curta duração seriam menos responsivas que as conexões de longa duração. Dessa forma, quando houvesse iteração entre fluxos de curta e longa duração, se esperava que as conexões de longa duração seriam penalizadas, obtendo uma banda passante efetiva reduzida. No entanto, os resultados obtidos em [13, 14] mostram uma situação diferente. Os autores observaram que os fluxos TCP de curta duração são responsivos, e em muitos casos mais responsivos que os fluxos TCP de longa duração. Basicamente, esse comportamento foi atribuído à natureza conservadora do controle de congestionamento do TCP.

Nessa seção é apresentado o resultado das simulações realizadas para comparar o desempenho, em termos de banda obtida, de conexões TCP de curta e longa duração. Na figura 4.2 é mostrado o desempenho das conexões TCP em função do tamanho médio de uma transferência (em bytes) para diferentes valores de carga total (ρ). Nos resultados apresentados na figura, a banda obtida é normalizada com relação a banda passante do enlace de gargalo. O tamanho das transferências (L) varia de acordo com uma distribuição exponencial de média L.

Os resultados são equivalentes aos obtidos em [13, 14]. Como pode ser visto, as curvas mostram que a banda passante obtida decresce com a diminuição do tamanho médio de uma transferência. Isso acontece porque conexões TCP de curta duração operam praticamente no período transiente do mecanismo de controle de congestionamento (fase *slow start*). Como já foi apresentado por outros autores,

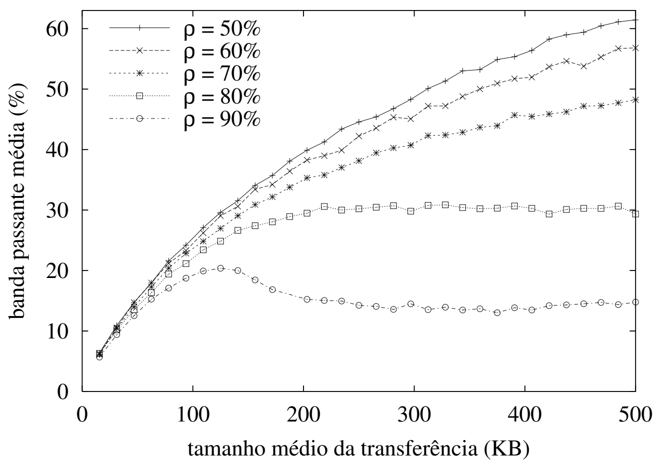


Figura 4.2: Banda passante obtida em função do tamanho médio de transferência.

as conseqüências são basicamente duas. Primeiro, o TCP não consegue, em alguns casos, consumir toda vazão que tem disponível. Segundo, durante a fase *slow start* o TCP é mais penalizado por perdas do que na fase *congestion avoidance*.

4.3 Gerenciamento Ativo de Filas

A experiência com a Internet já mostrou que o uso de mecanismos de controle de congestionamento são fundamentais para evitar que a rede entre em colapso sob determinadas condições de carga. Apesar desses mecanismos serem suficientes para trazer a rede a um estado útil, congestionamentos são fenômenos freqüentes que têm características indesejáveis como desperdício de recursos da rede e queda na função de utilidade observada pelos usuários. Além disso, os mecanismos de controle de congestionamento fim-a-fim, como o utilizado pelo TCP, fazem com que a rede passe por fases de baixa utilização. Dessa forma, é observada uma utilização média da rede abaixo do desejado. Um outro problema, ainda pouco freqüente, é o uso de protocolos de transporte sem controle de congestionamento fim-a-fim, como o UDP. Embora, atualmente, o número de fluxos que trafegam na Internet com esse tipo de transporte seja muito baixo, uma aplicação de larga escala como Telefonia IP poderia levar a rede a um colapso em pouco tempo.

Para tentar maximizar a utilização da rede e minimizar a freqüência com que ocorrem os congestionamentos, foram elaborados os mecanismos de gerenciamento ativo de fila capazes de detectar e sinalizar congestionamentos incipientes. O objetivo desses mecanismos é informar as entidades das extremidades que um congestionamento deve ocorrer caso o consumo de recursos continue a crescer. Atualmente, a perda ou a detecção de uma perda é uma notificação implícita de congestionamento para o TCP. Porém, existem outras propostas que sugerem o uso de informação no pacote, conhecidas como ECN, para notificar um congestionamento incipiente. Neste trabalho notificação/sinalização são usados de forma intercambiada com descarte.

Até o momento, o mecanismo de gerenciamento ativo de fila mais estudado é o RED, tornando-se candidato a substituto do *Drop Tail* desde a publicação de [15]. Já há no mercado equipamentos que têm o RED disponível. Diferente do RED, o *Drop Tail* é apenas uma fila do tipo FIFO que descarta o último pacote que chega à fila quando esta estiver cheia. De acordo com [55, 56, 57], o RED possui um conjunto de características que justificam a substituição do atual *Drop Tail*. Por outro lado, vários pesquisadores têm contestado tal substituição [58, 59, 60], alegando, sobretudo, que a complexidade de configuração e a sensibilidade dos parâmetros às características do tráfego não justificam seus benefícios. Outros vão mais longe, e descrevem situações nas quais o RED tem um desempenho igual ou inferior ao *Drop Tail*, e até mesmo que o RED não fornece algumas das vantagens alegadas.

As várias possibilidades oferecidas pelos mecanismos de gerenciamento ativo de filas motivou, e ainda motiva, estudos para melhorar o RED ou desenvolver novos mecanismos. Entre algumas modificações do RED estão o SRED [61], READ [62], ARED [63, 64] e DRED [65]. Alguns exemplos de propostas que apresentam novos paradigmas ao gerenciamento ativo de filas são BLUE [60], GREEN [66], REM [67], PI [68] e AVQ [69]. Em geral, esses trabalhos utilizam como referência para comparações o RED original.

A seguir é apresentada uma breve descrição do RED, o qual será posteriormente utilizado em algumas avaliações usando-se o modelo de tráfego HTTP de agregado.

4.3.1 *Random Early Detection*

O RED é um mecanismo de gerenciamento ativo de fila que é capaz de detectar congestionamentos incipientes e sinalizá-los através de notificação explícita (ECN) ou descarte. Para identificar esse tipo de congestionamento o RED monitora o tamanho médio da fila, iniciando a sinalização a partir de um limiar mínimo (min_{th}) com uma determinada probabilidade que cresce até o máximo (max_p) quando o tamanho médio atinge o limiar máximo (max_{th}). Após max_{th} o RED passa a sinalizar cada pacote que chega com probabilidade 1, ou seja, no caso do TCP atual todos os pacotes passam a ser descartados. Recentemente, passou a ser recomendado uma modificação nesse mecanismo conhecida como *gentle* [70] que utiliza outra probabilidade crescente a partir de max_{th} até 2*max_{th}. Essa modificação torna o RED mais robusto em relação a configuração dos parâmetros.

Além dos parâmetros citados (min_{th}, max_{th} e max_p), outro importante é w_q, que define o quão rápido o RED deve reagir a variações no tamanho médio da fila. [55] descreve a importância da configuração apropriada desses parâmetros e sugere alguns valores típicos para uma grande variedade de tráfego. Uma versão mais recente das configurações é apresentada em [71]. Embora não seja consenso, a maior parte dos trabalhos tem observado diferenças significativas no desempenho do RED ao configurar os parâmetros e que tal configuração é influenciada pelo perfil do tráfego ao qual o mecanismo está submetido.

O RED, ao ser desenvolvido, objetivou atingir algumas propriedades importantes, a saber:

- Capacidade de identificar congestionamentos incipientes e reagir apropriadamente. É importante para um mecanismo de controle de congestionamento manter o atraso médio da fila baixo ao mesmo tempo em que oferece uma alta vazão.

- Igualdade no tratamento de diferentes tipos de tráfego. Essa propriedade é importante sobretudo com relação a tráfego de rajada que é muito comum ao se utilizar protocolos de transporte como o TCP, sendo altamente penalizado por mecanismos como o *Drop Tail*.

- Impedimento de sincronização global. O efeito de sincronização global, observado em simulações, consiste na reação a congestionamento (quase) simultânea de todas as conexões de transporte que compartilham um mesmo enlace. Dessa forma, a utilização média do enlace cai de forma significativa, levando a um subutilização dos recursos e uma diminuição geral na função de utilidade dos usuários. O RED tenta evitar a sincronização global através do uso de descarte aleatório de pacotes.

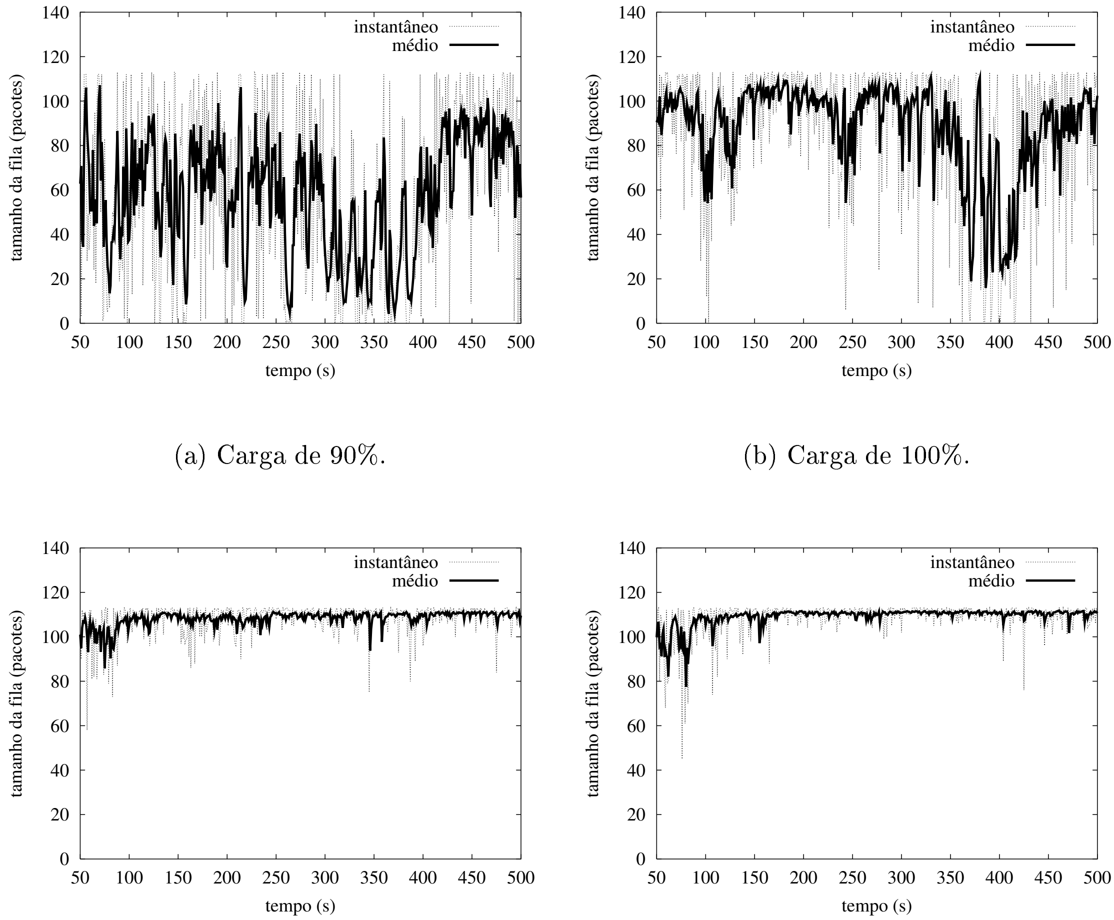
- Operação apropriada independente das fontes. Um mecanismo de gerenciamento ativo de fila deve ser capaz de controlar o tamanho médio da fila ainda que o protocolo de transporte utilizado não possua controle de congestionamento.

Atualmente, os defensores do RED sugerem o seu uso para fornecer baixo atraso médio na fila e alta vazão. Os mecanismos de fila são recomendados para enlaces de gargalo, sobretudo os que sofram congestionamentos freqüentes. Em redes de "melhor esforço", como a Internet, a utilização do RED ainda é alvo de discussões, mas em redes com qualidade de serviço ele pode ser usado em algumas situações de forma muito útil.

4.3.2 Simulações

Nas simulações foi utilizada a topologia apresentada na figura 4.1. O único gargalo da topologia é encontrado entre os nós $r2$ e $r3$, onde os mecanismos *Drop Tail* e RED são avaliados. Nessa topologia, a capacidade do enlace de acesso (M) determina a taxa de pico (ou taxa máxima) na qual rajadas emitidas pelas fontes chegam aos nós de agregação s_i.

De acordo com [37, 38], o tamanho do *buffer* escolhido para o enlace de gargalo é $2 * Bw * RTT$, onde Bw é a capacidade do enlace e RTT (*Round Trip Time*) é o valor do maior RTT de um fluxo que passe pelo enlace. Como nos experimentos realizados não havia interesse em analisar a influência do RTT no desempenho dos mecanismos, todos os fluxos experimentam o mesmo retardo, a menos da variação do tamanho da fila. O RED foi configurado de acordo as recomendações feitas em [71].

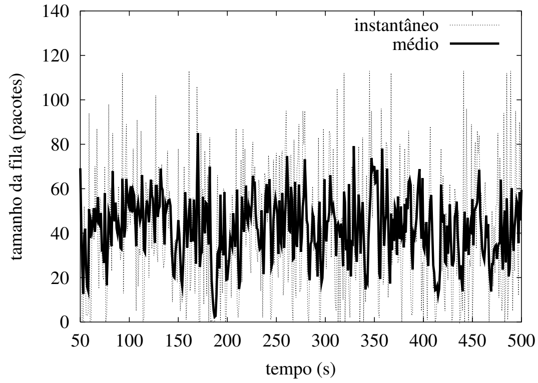


(a) Carga de 90%. (b) Carga de 100%.

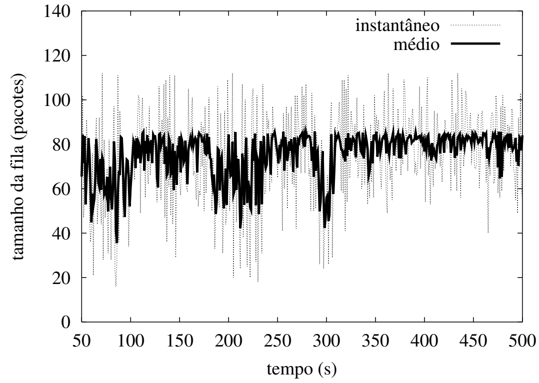
(c) Carga de 110%. (d) Carga de 120%.

Figura 4.3: Comportamento típico da fila utilizando *Drop Tail*.

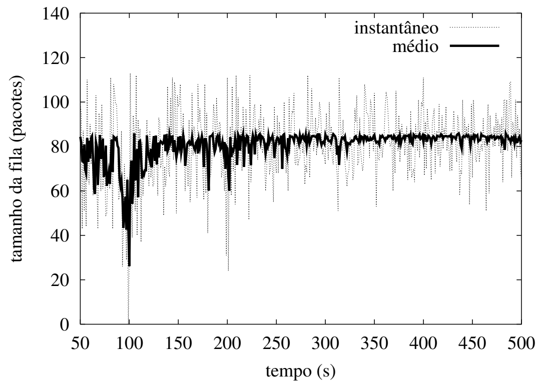
Três mecanismos foram escolhidos para avaliação: *Drop Tail*, RED e RED com o parâmetro *gentle* configurado (REDg). Conforme recomendado pelos desenvolvedores do RED [55, 64], as métricas "apropriadas" para avaliação do mecanismo devem ser baseadas no provedor de serviço, ou seja, na rede, em especial nos roteadores. Porém, métricas de interesse do usuário, tais como *goodput* e atraso, são importantes e podem ser obtidas facilmente a partir das de interesse do provedor



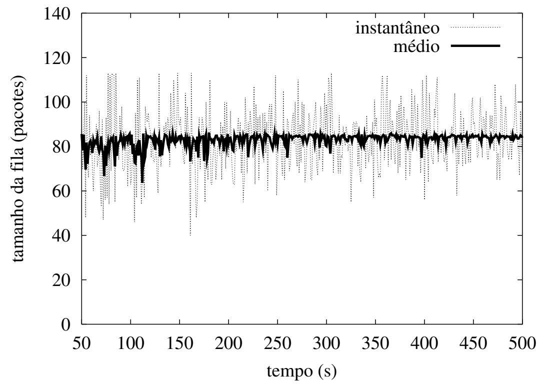
(a) Carga de 90%.



(b) Carga de 100%.



(c) Carga de 110%.

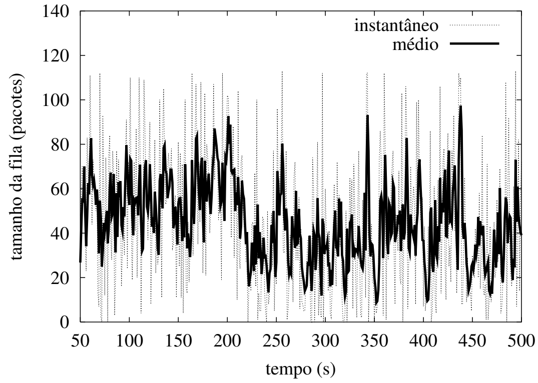


(d) Carga de 120%.

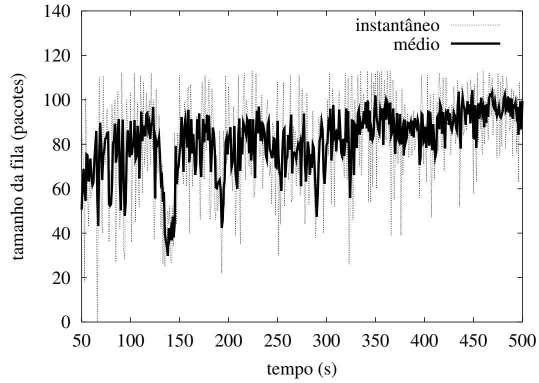
Figura 4.4: Comportamento típico da fila utilizando RED.

de serviço [1]. A escolha de tais métricas se justifica pela melhor percepção que se pode obter de um mecanismo de gerenciamento ativo de fila ao analisar informações do roteador. Dessa forma, as principais métricas apresentadas são vazão e taxa de perda. E de forma complementar, é mostrado ainda o tamanho instantâneo e médio da fila ao longo do tempo, com os três mecanismos sob avaliação. A maior parte das avaliações do RED utilizam fontes de tráfego de longa duração, porém, como já foi mostrado, a maior parte do tráfego da Internet é HTTP. Nas avaliações foi preferida a abordagem utilizada em [72, 73], onde os autores usam fontes de tráfego HTTP, porém utilizando o modelo HTTP de agregado proposto nesse trabalho. A

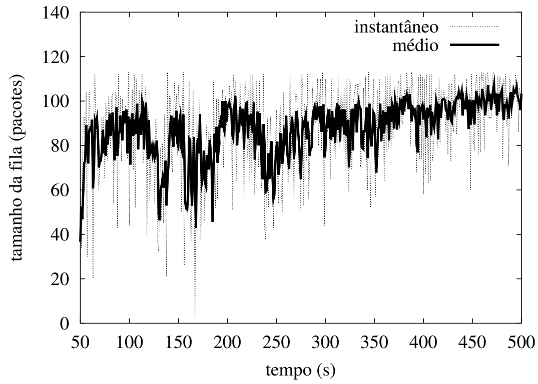
[1] Nesse contexto, essas métricas são também chamadas de métricas do roteador.



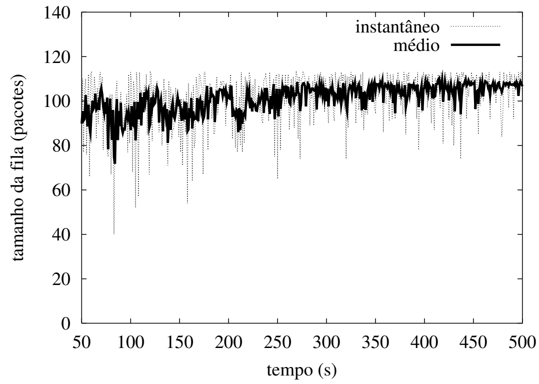
(a) Carga de 90%.



(b) Carga de 100%.



(c) Carga de 110%.



(d) Carga de 120%.

Figura 4.5: Comportamento típico da fila utilizando RED com o parâmetro *gentle* configurado.

carga aplicada na rede varia de 90% a 120%, no intuito de representar enlaces de gargalo congestionados.

As figuras 4.3 a 4.5 mostram o comportamento típico da fila sob diferentes cargas e utilizando *Drop Tail*, RED e REDg. Conforme esperado, ao utilizar o *Drop Tail*, a fila tende a permanecer completamente ocupada à medida que a carga aumenta. O RED mantém uma restrição rígida do tamanho médio, uma vez que a partir de max_{th} a probabilidade de descarte passa ser igual 1. Dessa forma, é possível diminuir o atraso médio observado pelas aplicações. Por outro lado, conforme foi mostrado em [55, 64], o RED gera uma oscilação significativa na fila, aumentando a variação do atraso. O REDg oferece uma restrição mais relaxada do atraso médio da fila e apresenta menor oscilação que o RED. Os mecanismos RED e REDg foram configurados de forma semelhante para compatibilizar as comparações, porém os resultados sugerem que REDg pode apresentar um desempenho melhor que o RED com outra configuração dos parâmetros. Apesar de um estudo extensivo sobre configuração do REDg estar fora do escopo deste trabalho, esse pode ser mais um uso para o modelo proposto.

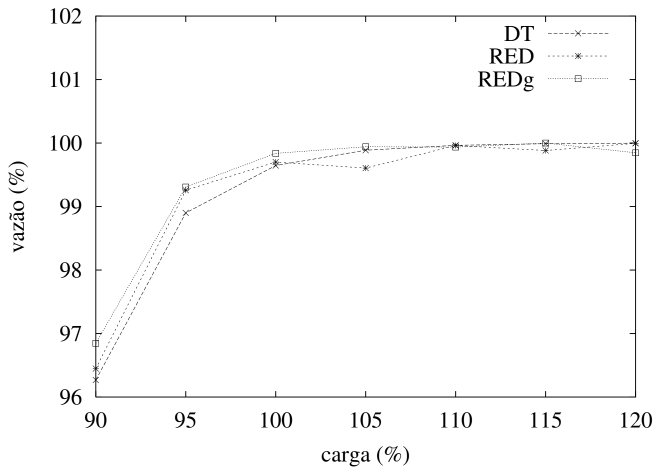


Figura 4.6: Percentagem consumida da vazão do enlace de gargalo à medida que a carga aumenta.

A figura 4.6 mostra o percentual médio da vazão à medida que a carga aumenta. Como pode ser visto, não há diferença significativa de desempenho entre os mecanismos. Ainda que de forma sutil, é possível observar um comportamento mais estável dos mecanismos *Drop Tail* e REDg em relação ao RED. Esses resultados confirmam algumas observações feitas em [59, 73], que basicamente descrevem não haver ganho de desempenho no consumo de banda passante ao se substituir o *Drop Tail* por RED. Como foi observado, o REDg também não apresenta melhorias nesse sentido.

Na figura 4.7 é exibida a taxa de perda de pacotes com o aumento da carga, para cada um dos mecanismos avaliados. Com cargas inferiores a 100%, o *Drop Tail* possui uma taxa de erro menor que o RED e o REDg. Esse comportamento é esperado, pois o *Drop Tail* somente realiza descartes quando o *buffer* transborda,

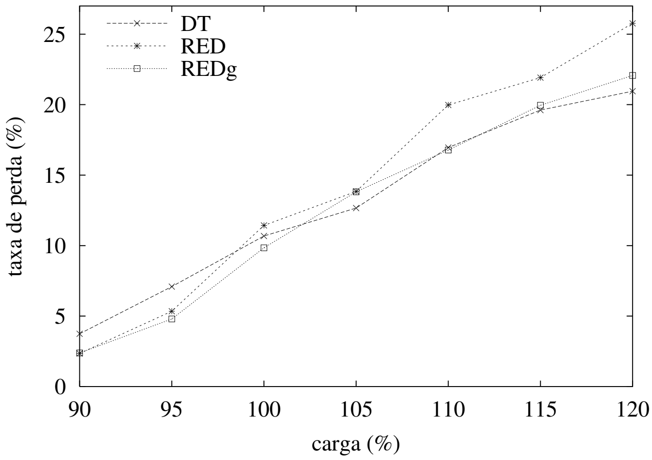


Figura 4.7: Taxa de perda no enlace de gargalo à medida que a carga aumenta.

sendo que a probabilidade de ocorrer esse fato é diretamente proporcional à carga. Com cargas a partir de 100%, *Drop Tail* e REDg passam a ter taxas de descarte semelhantes, enquanto o RED é mais agressivo. Isso ocorre porque o RED mantém o tamanho médio da fila sob controle restrito através do descarte de pacotes, o qual precisa ser mais intenso à medida que a carga aumenta.

4.4 Descarte Seletivo de Pacotes e Políticas de *Push-out*

O modelo de serviços diferenciados na Internet (DiffServ) [74] tem recebido grande atenção por parte da comunidade científica. Um dos mecanismos de controle de tráfego, nesse modelo, é o descarte seletivo usado para discriminar pacotes pertencentes a diferentes classes de serviço. O mecanismo de descarte seletivo descarta pacotes em situações de congestionamento de acordo com o nível de prioridade dos mesmos. A perda de pacotes devido a descarte pode ser tanto devido a políticas de gerenciamento de *buffer*, quanto a políticas de escalonamento de pacotes. O descarte seletivo discrimina pacotes sem desordená-los, podendo ser usado no tratamento de pacotes do serviço Assegurado [75], definido no modelo DiffServ. Além disso, pela característica de não desordenamento, essas disciplinas podem ser aplicadas na diferenciação de pacotes pertencentes a um mesmo fluxo.

As políticas de descarte seletivo têm uma influência significativa no desempenho obtido por clientes de diferentes classes de serviço. Esse comportamento é destacado quando esses clientes são sensíveis a perdas, como é o caso dos fluxos TCP. Para esse tipo de cliente, as perdas podem ser usadas para determinar a alocação da banda passante dos enlaces de gargalo. Dentre desse contexto, um mecanismo que tem despertado interesse ao lidar com múltiplas classes de usuários é o RED-n (RED com múltiplos níveis de prioridade) ou RED Multiclasse [76]. Há ainda uma versão comercial do RED-n conhecida como WRED [77]. O RED-n é um mecanismo de gerenciamento de *buffer* que sinaliza quando um descarte deve ser realizado, porém não determina qual pacote deve ser descartado. Por questões de facilidade de implementação, o pacote que chega é o candidato a descarte. Uma outra possibilidade de se definir políticas de descarte é permitir que pacotes já enfileirados sejam removidos ao invés de se descartar pacotes no momento de sua chegada. Tal política é chamada de *push-out*. Até recentemente, as políticas de remoção (*push-out*) eram consideradas de difícil implementação por implicarem numa reorganização da fila. No entanto, com as novas tecnologias usadas no projeto de comutadores de alta velocidade, baseadas em ASICs, constata-se que uma grande parte das políticas de remoção pode ser implementada sem impacto significativo no desempenho [78, 79].

Esta seção tem como objetivo comparar políticas de descarte em termos da diferenciação fornecida, usando o tráfego HTTP agregado (dominado por fluxos TCP de curta duração) e fluxos TCP de longa duração pertencentes a classes de serviço distintas. É realizada uma comparação da eficiência do uso de mecanismos baseados apenas em *push-out* em relação a outros que não descartam pacotes enfileirados (RED-n). Também é investigado o uso integrado de *push-out* com RED-n.

4.4.1 Descarte seletivo de pacotes

Em redes ATM, as políticas de descarte de pacotes foram concebidas para contornar o problema de baixo desempenho observado devido a fragmentação dos pacotes em células, onde a perda de apenas uma das células de um pacote exige a retransmissão de todas elas. Assim, várias políticas de descarte [80, 81, 82, 83, 84] foram projetadas para aliviar o congestionamento e aumentar a utilização da rede. Essa políticas são importantes porque quando uma célula de um pacote for perdida (ou corrompida), todas as células seguintes do mesmo pacote devem ser selecionadas para descarte. Os objetivos são maximizar a utilização e melhorar o desempenho das aplicações, tentando atingir uma "melhor" taxa de perda de mensagens através do agrupamento das células perdidas.

Em redes IP, as políticas de descarte funcionam de forma diferente. O problema de fragmentação não é uma questão importante e os objetivos são outros. As políticas de descarte de pacotes para redes IP têm se concentrado na redução do comprimento das filas, de forma a diminuir o atraso de pacotes com requisitos de tempo-real, evitar problemas de sincronização global do TCP [55] e evitar que um ou mais fluxos monopolizem o espaço de *buffer* (*lock-out*) [15]. Mais recentemente, as políticas de descarte têm sido usadas para fornecer algum tipo de garantia através da discriminação dos pacotes.

Descarte seletivo de pacotes engloba dois tipos de políticas: as políticas de gerenciamento de *buffers*, que determinam quando algum pacote deve ser descartado, e as políticas de remoção, que determinam qual pacote deve ser descartado, incluindo não apenas aquele que chega, mas todos os pacotes em espera para transmissão.

Políticas de gerenciamento de *buffers*

No Serviço Assegurado, o gerenciamento de *buffers* é, em geral, realizado pela adoção de algoritmos RED: um para cada nível de preferência de descarte. Cada algoritmo RED é configurado de maneira mais agressiva que aquele que trata dos pacotes do nível de preferência exatamente acima, visando o descarte prioritário de pacotes com um maior valor de preferência de descarte. O objetivo é reduzir os efeitos do congestionamento antes da necessidade de descarte de pacotes de menor preferência. Este mecanismo é conhecido como RED-n (RED com múltiplos níveis de prioridade de descarte). Em [76], os autores apresentam quatro categorias gerais para políticas RED quando múltiplos níveis de preferência de descarte, ou múltiplas cores, são utilizados na marcação de pacotes. Essas categorias surgem da maneira pela qual se pode calcular o tamanho médio da fila e estabelecer os limiares de descarte do algoritmo RED.

Na categoria Múltiplas Médias/Múltiplos Limiares, um tamanho médio da fila é calculado para cada nível de preferência, onde o número de pacotes na fila de um determinado nível é igual à soma dos pacotes desse nível e, se existirem, dos níveis inferiores. Além disso, para cada nível de preferência de descarte tem-se diferentes limiares de descarte associados. Por exemplo, a fila RIO (RED com *IN* e *OUT*) [85] pertence a essa categoria. O tamanho médio da fila para pacotes *IN* (alta prioridade) é calculado utilizando apenas o número de pacotes *IN*, enquanto que o tamanho médio da fila para pacotes *OUT* (baixa prioridade) é calculado utilizando o número de pacotes *IN+OUT*. Diferentes limiares de descarte são definidos para cada um destes níveis. Com essa categoria, é possível ser tão agressivo quanto se deseje no descarte de pacotes com maiores valores de precedência de descarte. Por questões de simplicidade de implementação, o pacote a ser descartado é sempre aquele que chega.

Políticas de remoção

Ao considerar os mecanismos de remoção de pacotes da fila, diferentes políticas podem ser aplicadas: *First-In-First-Drop* (FIFD), *Last-In-First-Drop* (LIFD), *Modified-FIFD* (M-FIFD) e *Random* (RAND). Tipicamente, a discussão de qual política usar ocorre quando há mais de uma classe de tráfego compartilhando um *buffer* ou pacotes com diferentes prioridades.

As políticas de remoção determinam qual pacote será descartado quando um pacote de mais alta prioridade chega e o gerenciamento de *buffer* determina que um pacote deve ser descartado. Nessa situação, o gerenciamento de fila é, em geral, apenas uma fila FIFO que responde à situação de não poder aceitar o pacote que chega a não ser que algum outro da fila seja removido.

Na política FIFD, o pacote de menor prioridade mais próximo à cabeça da fila será descartado. Na política LIFD, o pacote de menor prioridade mais próximo do final da fila será descartado. A escolha de uma dessas políticas tem implicações na distribuição das perdas e, conseqüentemente, no desempenho alcançado. O descarte mais próximo à cabeça da fila faz com que o receptor detecte uma perda mais rapidamente, além de aumentar a probabilidade que um pacote com menor prioridade seja encontrado na fila ao chegar um pacote de maior prioridade [86]. Desta forma, a política FIFD fornece uma menor taxa de perda para pacotes de maior prioridade. Outras políticas, tais como M-FIFD, permitem que um pacote de menor prioridade retire um pacote de mesma classe já enfileirado. Esse tipo de política aumenta ainda mais a probabilidade de um pacote com menor prioridade ser encontrado na fila na chegada de um próximo pacote de mais alta prioridade. A política RAND escolhe aleatoriamente um pacote de menor prioridade a ser descartado. Não existe nenhuma vantagem aparente na utilização dessa política. Foram realizadas simulações com as políticas FIFD, M-FIFD, LIFD e RAND, a fim de comparar o seu desempenho em relação à banda passante, taxa de perda e atraso na fila. Sobretudo para conexões de curta duração, os resultados obtidos mostraram que não há diferença significativa entre as políticas.

4.4.2 Simulações

Nesta seção, três diferentes mecanismos de descarte seletivo são avaliados e comparados: PO (*complete sharing push-out*), RIO e PRIO (*push-out* RIO). O primeiro mecanismo (PO) utiliza uma política *loss conserving*. Essa política utiliza o máximo possível dos *buffers* disponíveis, evitando descartar quaisquer pacotes antes do transbordo. O RIO funciona conforme descrito anteriormente. Por fim, o PRIO utiliza o RIO para sinalizar a necessidade de descarte de um pacote, mas faz uso da política de remoção FIFD para dar lugar ao pacote que chega. Os dois últimos mecanismos (RIO e PRIO) são do tipo *non-loss conserving*.

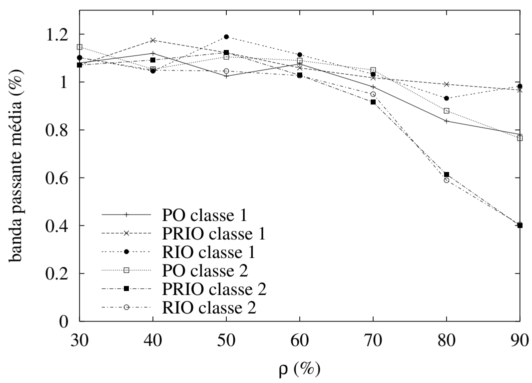
Para comparar esses mecanismos e suas políticas, foram utilizadas métricas de desempenho que capturam o ponto de vista tanto do provedor do serviço como do usuário. Do ponto de vista do provedor, a primeira métrica é a eficiência no aproveitamento da banda passante, ou seja, a utilização da banda passante. Dada uma certa carga na rede, uma baixa utilização do enlace pode ser inaceitável. Além disso, se o provedor do serviço utiliza uma tarifação baseada na banda passante consumida pelo cliente, ao invés de uma tarifação fixa (*flat rate*), seria desejável otimizar essa métrica de desempenho, o que aumentaria a sua arrecadação. Uma outra métrica importante é a taxa de perda de pacotes. A perda de pacotes representa banda passante e *buffers* desperdiçados nos enlaces anteriores. Para o usuário, a métrica escolhida o grau de diferenciação fornecido pelos mecanismos avaliados. Ela corresponde à razão entre as taxas obtidas por uma classe de serviço e pela classe exatamente inferior, o que representa o espaçamento entre essas classes. O usuário que utiliza uma classe superior, e paga mais por isso, deseja obter um serviço de melhor qualidade do que aquele que paga menos. No entanto, isso não significa que se deve ignorar o desempenho oferecido aos pacotes de menor prioridade, uma vez que os pacotes de mais alta prioridade estejam protegidos.

Novamente a topologia utilizada é a da figura 4.1. O enlace de gargalo corresponde ao enlace entre os nós $r1$ e $r2$. Os nós $r1$ e $r2$ implementam os seguintes mecanismos de gerenciamento de filas: PO, RIO e PRIO. Este enlace, de capacidade C, é compartilhado por duas classes de tráfego. A taxa de chegada de clientes da classe i ($i \in 1, 2$) é dado por λ_i, ou seja, a cada $1/\lambda_i$ chega um novo microfluxo da classe i por um enlace de capacidade M de acesso ao nó s_i. Logo após o nó de agregação s_i, o tráfego originado dos vários clientes de uma mesma classe representa uma carga ρ_i. Esta carga atravessa o enlace de gargalo em direção ao nó d_i. A condição de estabilidade do sistema é dada por $\rho = \rho_1 + \rho_2 < 1$.

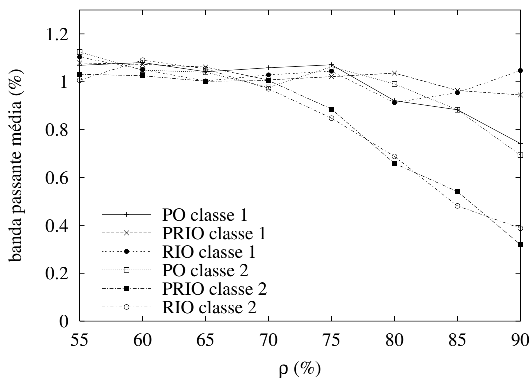
Tráfego HTTP

Inicialmente, foram realizadas simulações com o modelo de tráfego HTTP de agregado e os três tipos de políticas de descarte seletivo selecionados. É importante lembrar, que nas distribuições utilizadas no modelo houve um predomínio de conexões de curta duração. Nestas simulações a carga da classe 1 é mantida fixa enquanto a carga total aumenta até 90%. Basicamente, o objetivo destas simulações foi identificar a sensibilidade dos mecanismos com relação ao aumento da carga. Um resultado desejável seria que o desempenho da classe mais prioritária (classe 1) permanecesse constante com o aumento da carga total. Ou seja, que o mecanismo de descarte seletivo fornecesse um isolamento adequado entre as classes.

As figuras 4.8(a) e 4.8(b) apresentam os gráficos da banda passante média obtida pelos clientes das classes 1 e 2 com o aumento da carga, para ρ_1 (carga da classe 1) igual a 20% e 50%, respectivamente. Outros valores foram avaliados e produziram resultados semelhantes aos apresentados aqui. Novamente, a banda obtida por cada classe é normalizada com relação a banda passante do enlace de gargalo.



(a) $\rho_1 = 20\%$



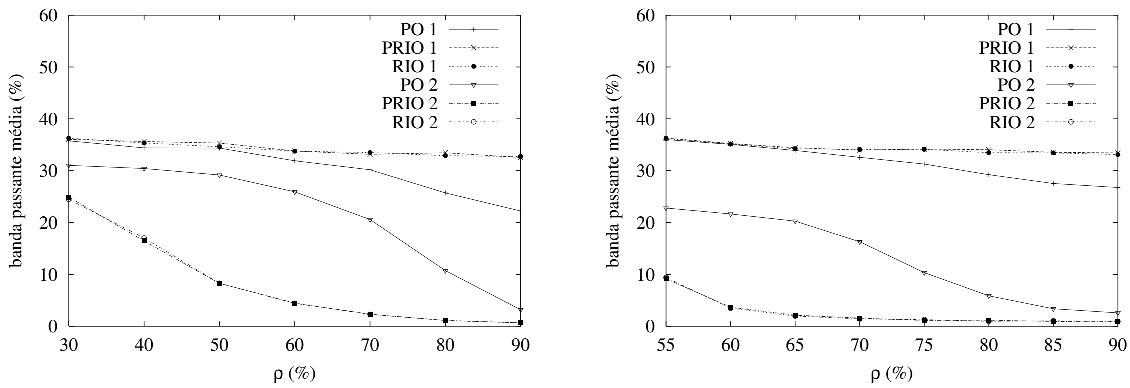
(b) $\rho_1 = 50\%$

Figura 4.8: Banda passante das classes 1 e 2 em função da carga na rede.

Como pode ser observado nas figuras 4.8(a) e 4.8(b), o desempenho permanece quase constante com o aumento da carga para as políticas PRIO e RIO. Uma outra constatação é que o mecanismo de remoção (*push-out*) da disciplina PRIO não oferece ganho no desempenho, tanto para a classe 1 quanto para a classe 2. O mecanismo mais sensível ao aumento da carga é o PO. Além disso, essa política, apesar de oferecer um melhor desempenho para a classe 2, não diferencia significativamente as classes. Pode ser observado ainda uma queda brusca na banda passante da classe 2 para as políticas PRIO e RIO, a medida que a carga total aumenta. Isso faz com que a banda média obtida por tais clientes seja muito baixa, o que poderia ser explicado pelo fato das conexões serem em sua maioria de curta duração. No intuito de compreender melhor o comportamento dos mecanismos e realizar uma comparação mais aprofundada entre as políticas de descarte seletivo, foram realizadas simulações com conexões de longa duração. Os resultados obtidos são apresentados a seguir.

Conexões de longa duração

Em seguida, foram realizadas simulações utilizando o modelo de tráfego com uma configuração diferente das usadas até o momento. O tamanho das transferências foi descrito através de uma distribuição exponencial com média de 512 Kbytes, ou seja, as conexões passaram a ser predominantemente de longa duração. As cargas impostas por cada classe são distribuídas da mesma forma que nas simulações da seção anterior. As figuras 4.9(a) e 4.9(b) mostram a banda passante média obtida pelos clientes das classes 1 e 2 com o aumento da carga, para ρ_1 (carga da classe 1) igual a 20% e 50%, respectivamente.

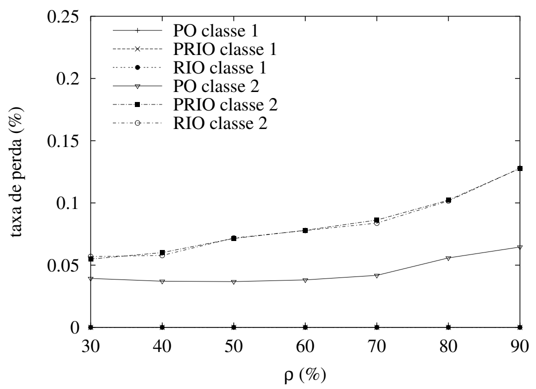


(a) $\rho_1 = 20\%$ (b) $\rho_1 = 50\%$

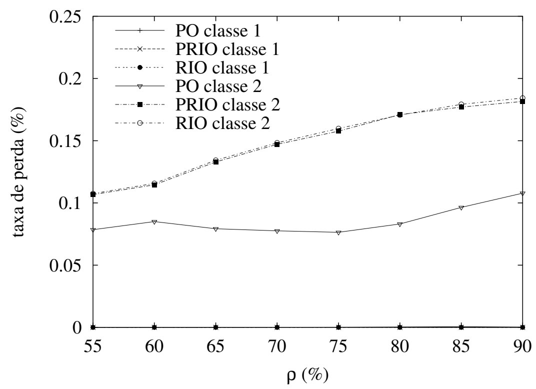
Figura 4.9: Banda passante da classes 1 e 2 em função da carga na rede.

Uma vez que existiam diferenças nos resultados em comparação aos obtidos com conexões de curta duração, sobretudo com relação à política PO, decidiu-se avaliar outras métricas com o intuito de compreender melhor o comportamento observado. Nesse sentido, para analisar os resultados em termos de banda passante média obtida escolheu-se avaliar métricas de desempenho que têm um forte impacto no desempenho do protocolo TCP. Essas métricas são: taxa de perda, número médio de temporizações e atraso. Quanto maiores forem os valores obtidos nessas métricas pior é o desempenho do protocolo TCP. Os gráficos das figuras 4.10 a 4.12 mostram os resultados dessas avaliações, para ambas as classes, em função do aumento da carga na rede.

Como pode ser observado, as políticas de descarte baseadas em RED (RIO e PRIO) oferecem maior banda passante para a classe mais prioritária. Esse fato se deve principalmente ao aumento significativo da taxa de perda da classe menos prioritária (figuras 4.10(a) e 4.10(b)), característico das políticas de descarte prévio (*early discard*) com múltiplas classes. No entanto, ocorre uma pequena queda na banda passante da classe 1 devido a um pequeno aumento do retardo, uma vez que não ocorrem perdas e nem temporizações (figuras 4.12(a) e 4.12(b)).



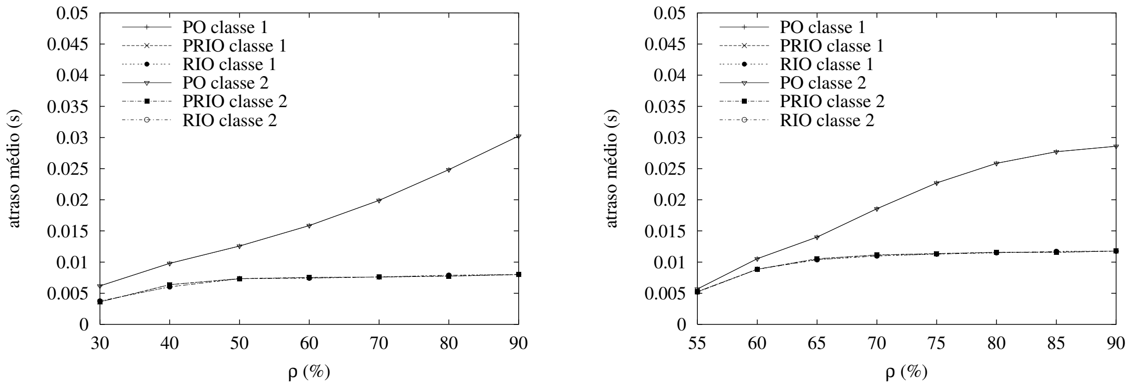
(a) $\rho_1 = 20\%$



(b) $\rho_1 = 50\%$

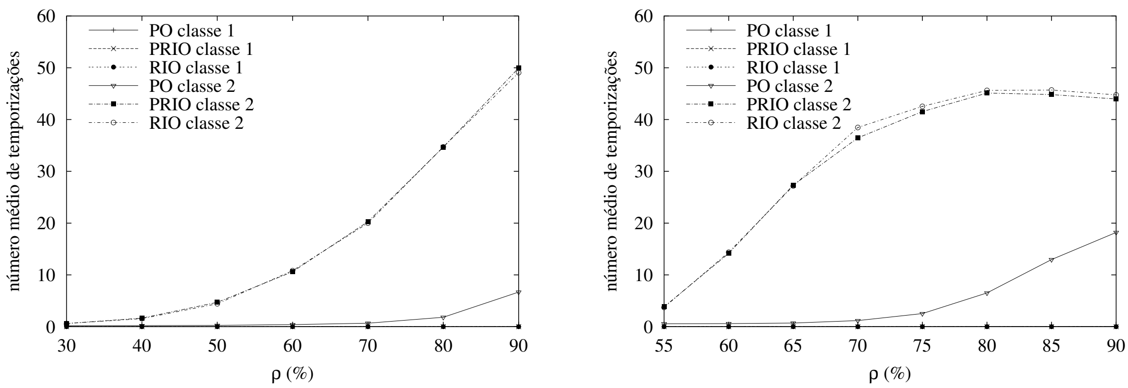
Figura 4.10: Taxa de perda das classes 1 e 2 em função da carga na rede.

A partir das figuras 4.9(a) e 4.9(b) é possível ainda observar que a política PO não diferencia drasticamente as classes, pois a diferença entre as taxas de perda das classes permanece menor que as apresentadas pelo RIO e pelo PRIO, conforme observado nas figuras 4.10(a) e 4.10(b). Isso ocorre porque essa política somente efetua descartes com o transbordo do *buffer* do enlace de gargalo. A diferenciação de classes do PO não é observada nas figuras 4.8(a) e 4.8(b) porque o tráfego de conexões de curta duração não permite que o TCP atinja seu estado estacionário e, portanto a banda passante obtida por cada fluxo é muito baixa. Como o descarte ocorre somente quando há transbordo do *buffer*, a política PO não penaliza como



(a) $\rho_1 = 20\%$ (b) $\rho_1 = 50\%$

Figura 4.11: Atraso na fila em função da carga na rede.



(a) $\rho_1 = 20\%$ (b) $\rho_1 = 50\%$

Figura 4.12: Número médio de temporizações das classes 1 e 2 em função da carga na rede.

deveria a classe menos prioritária.

A maior queda na banda de ambas as classes para a política PO se deve ao aumento significativo do atraso na fila de acordo com o gráfico das figuras 4.11(a) e 4.11(b). Por outro lado, as políticas PRIO e RIO mantém um baixo atraso nas filas, característico de políticas de descarte prévio.

Além disso, se confirmou a observação que RIO e PRIO não apresentam diferenças significativas de desempenho. Um dos motivos para esse fato é que ambos são baseados em um mecanismo de descarte prévio que sobrepõem a função principal das políticas de remoção: liberar espaço de um pacote mais antigo para dar lugar a um mais recente. A diferença é que mecanismos com descarte prévio (como RIO) já realizam essa tarefa diminuindo o atraso das filas. Outro motivo é que a característica das políticas de remoção de poder alterar a distribuição das perdas se torna insignificante quando combinada com um mecanismo de descarte prévio, porque a ocupação média do *buffer* é mantida baixa e o descarte dos pacotes OUT é agressivo. Dessa forma, a influência dos pacotes OUT, em pequena quantidade, se torna negligenciável.

Capítulo 5

Conclusão e Perspectivas para Trabalhos Futuros

DEVIDO a ampla utilização da *Web*, o protocolo HTTP é, atualmente, responsável pela maior parte dos bytes transmitidos na Internet. Logo, se torna importante estudar e avaliar esse tipo de tráfego. Para tanto, modelagem tem sido uma ferramenta largamente utilizada. Nesse trabalho foi mostrado o desenvolvimento e avaliação de um modelo tráfego de agregado e alguns exemplos de sua utilização.

Como foi apresentado, o primeiro passo para o estudo de tráfego é a coleta e análise do mesmo. Para realizar essa tarefa o método mais utilizado é o de registro de pacotes, que ao coletar o tráfego HTTP tem exigido o desenvolvimento de heurísticas e ferramentas. Após a coleta, as informações podem ser usadas no desenvolvimento de diferentes tipos de modelos. Um tipo de modelo amplamente utilizado é o analítico paramétrico que possui um conjunto de parâmetros descritos através de distribuições de probabilidade. Uma fase importante para esse tipo de modelo é o estudo dos dados coletados em busca das distribuições que melhor descrevem cada parâmetro. Vários trabalhos foram realizados nessas áreas e há disponível na literatura muitos resultados. Nesse sentido, o presente trabalho optou por fazer uso dos dados disponíveis para a construção do modelo. No entanto, dado o tipo do modelo, caso exista necessidade de modificar a configuração dos parâmetros devido a novos resultados em coleta e avaliação do tráfego na rede, esta pode ser feita de forma simples.

O modelo de agregado proposto nesse trabalho visa a simplicidade pelo número reduzido de parâmetros, porém considerando a importância da acurácia na representação do tráfego. Além disso, o modelo oferece um controle simplificado da carga imposta à rede, o qual se acredita ser um recurso útil em vários tipos de avaliações. Foi mostrado o desenvolvimento e avaliação do modelo e como ele cumpre as propostas de projeto. Dentre as características úteis observadas durante o desenvolvimento do modelo, vale destacar:

- controle da carga na rede utilizando um número reduzido de parâmetros;
- capacidade de gerar diferente número de conexões simultâneas;
- facilidade de representar e gerar cargas acima de 100% na rede;
- reprodução da característica de auto-similaridade.

Além do desenvolvimento do modelo de tráfego, foram realizados estudos de caso de aplicação do mesmo, a saber:

- controle de tráfego do TCP para conexões de curta e longa duração;
- mecanismos de gerenciamento ativo de filas;
- descarte seletivo de pacotes.

No último estudo de caso, as avaliações não se limitaram aos mecanismos existentes e seus usos tradicionais. Foi introduzido o uso de políticas de remoção *push-out* para realizar diferenciação de classes de serviço dentro da arquitetura DiffServ. Além disso, foi estudado o resultado da associação das políticas de *push-out* com mecanismos de gerenciamento de *buffer* de descarte prévio.

Além do que foi mostrado, o modelo proposto pode ser utilizado em outras situações, conforme foi ilustrado no texto. Na verdade, o modelo é facilmente extensível para reproduzir não somente o tráfego HTTP, mas outros tipos que possam ter suas distribuições de tamanho de transferência capturadas e utilizem o protocolo TCP como transporte.

O presente trabalho apresenta algumas perspectivas a serem cumpridas:

- implementação do gerador de tráfego sobre a plataforma de código aberto Linux utilizando *sockets*, já em adiantado estágio de desenvolvimento;
- elaboração de mais exemplos de uso do modelo e comparação de resultados de simulação do NS com a implementação em Linux;
- disponibilização da ferramenta, documentação e resultados para a comunidade científica através da *Web*, sendo que parte já está disponível;
- realizar testes e avaliações dentro da infra-estrutura da RNP2.

Referências Bibliográficas

[1] CAIDA (Cooperative Association for Internet Data Analysis) - Characterization of Internet traffic loads, segregated by application. http://www.caida.org/analysis/workload/byapplication/.

[2] Internet 2. http://www.internet2.org.

[3] Mah, B. An empirical model of http network traffic. In *Proc. INFOCOM'97* (abril de 1997).

[4] Molina, M., Castelli, P., and Foddis, G. Web Traffic Modeling Exploiting TCP Connections' Temporal Clustering through HTML-REDUCE. *IEEE Network Magazine 14*, 3 (2000), 46–55.

[5] Barford, P., and Crovella, M. Generating representative web workloads for network and server performance evaluation. In *Proc. ACM SIGMETRICS Conference* (julho de 1998), pp. 151–160.

[6] Abrahamsson, H., and Ahlgren, B. Using Empirical Distributions to Characterize Web Client Traffic and to Generate Synthetic Traffic. In *IEEE/Globecomm'00* (San Francisco, novembro de 2000).

[7] Choi, H.-K., and Limb, J. O. A Behavioural Model of Web Traffic. In *International Conference of Networking Protocol 99 (ICNP99)* (1999).

[8] Floyd, S., and Paxson, V. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking 9*, 4 (agosto de 2001), 392–403.

[9] Network Simulator (NS). http://www.isi.edu/nsnam/.

[10] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol – HTTP/1.1. *Internet RFC 2616* (abril de 1999).

[11] Berners-Lee, T., Fielding, R., and Frystyk, H. Hypertext Transfer Protocol – HTTP/1.0. *Internet RFC 1945* (maio de 1996).

[12] Krishnamurthy, B., Mogul, J. C., and Kristol, D. M. Key Differences between HTTP/1.0 and HTTP/1.1. Relatório técnico, ATT Labs, dezembro de 1998.

[13] Mahdavi, J. Mice vs. elephants: Facts and myths about small transfers on the internet. Relatório técnico, Novell, 1999.

[14] Guo, L., and Matta, I. The War between Mice and Elephants. In *Proc. ICNP'2001: The 9th IEEE International Conference on Network Protocols* (Riverside, 2001).

[15] Braden, B., Clark, D. D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K. K., Shenker, S., Wroclawski, J., and Zhang, L. Recommendations on queue management and congestion avoidance in the Internet. *Internet RFC 2309* (abril de 1998).

[16] Heidemann, J. Performance Interactions between P-HTTP and TCP implementations. *ACM Computer Communication Review 27*, 2 (abril de 1997), 65–73.

[17] Deng, S. Empirical model of WWW document arrivals at access link. In *ICC - International Communication Conference* (1996).

[18] Casilari, E., Reyes, A., Gonzalez, F. J., Estrella, A. D., and Sandoval, F. Characterisation of Web Traffic. In *Internet Performance Symposium* (San Antonio, novembro de 2001).

[19] Catledge, L. D., and Pitkow, J. E. Characterizing browsing strategies in the World-Wide Web. In *Third International World Wide Web Conference* (San Antonio, abril de 1995).

[20] Cunha, C. R., Bestavros, A., and Crovella, M. E. Characteristics of WWW client-based traces. Relatório técnico, Boston University, julho de 1995.

[21] Bell Labs Internet Traffic Research.
`http://cm.bell-labs.com/cm/ms/departments/sia/InternetTraffic/`.

[22] Sprint ATL IP Monitoring Project.
`http://www.sprintlabs.com/Department/IP-Interworking/Monitor/`.

[23] Feldmann, A. BLT: Bi-layer tracing of HTTP and TCP/IP. *WWW9 / Computer Networks 33*, 1-6 (2000), 321–335.

[24] Park, K., Kim, G., and Crovella, M. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proc. IEEE International Conference on Network Protocols* (outubro de 1996), pp. 171–180.

[25] Crovella, M. E., and Bestavros, A. Self-similarity in World Wide Web traffic: Evidence and possible causes. In *Proc. of the ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems* (Philadelphia, 1996), pp. 160–169.

[26] Barford, P., Bestavros, A., Bradley, A., and Crovella, M. Changes in Web Client Access patterns: Characteristics and Caching Implications. In *Special Issue on Characterization and Performance Evaluation* (1999).

[27] Reyes-Lecuona, A., González-Parada, E., Casilari, E., and Díaz-Estrella, A. A page-oriented WWW traffic model for wireless system simulations. In *Proc. of the 16th International Teletraffic Congress (ITC'16)* (junho de 1999).

[28] Floyd, S., and Henderson, T. The newreno modification to tcp's fast recovery algorithm. *Internet RFC 2582* (abril de 1999).

[29] Floyd, S., Mahdavi, J., Mathis, M., and Podolsky, M. An Extension to the Selective Acknowledgement (SACK) Option for TCP. *Internet RFC 2883* (julho de 2000).

[30] Fall, K., and Floyd, S. Simulation-based Comparisons of Tahoe, Reno and SACK TCP. *Computer Communication Review 26*, 3 (julho de 1996), 5–21.

[31] Jacobson, V. Congestion avoidance and control. In *ACM SIGCOMM '88* (Stanford, agosto de 1988), pp. 314–329.

[32] Mogul, J. C. Observing TCP Dynamics in Real Networks. *Proc. SIGCOMM '92 Symposium on Communications Architectures and Protocols* (1992), 305–317.

[33] M. Allman, V. Paxson, W. S. TCP congestion control. *RFC 2581* (abril de 1999).

[34] dos Santos, G. P. T., and de Vasconcellos, S. V. Avaliação por Simulação dos Mecanismos de Controle de Congestionamento do TCP. Relatório técnico, DEL/EE/UFRJ, agosto de 2000.

[35] Cardoso, K. V., and de Rezende, J. F. HTTP Traffic Modeling: Development and Application. In *International Telecommunications Symposium (ITS'2002)* (setembro de 2002).

[36] Kleinrock, L. *Queueing Systems.* ISBN 0471491101. John Wiley and Sons Inc., 1975.

[37] IRTF end2end-interest mailing list archive. `ftp://ftp.isi.edu/end2end/end2end-interest-1998.mail`.

[38] Villamizar, C., and Song, C. High Performance TCP in ANSNET. *Computer Communication Review 24*, 5 (outubro de 1995), 45–60.

[39] CAIDA (Cooperative Association for Internet Data Analysis) - Packet Sizes and Sequencing. `http://www.caida.org/outreach/resources/learn/packetsizes/`.

[40] Leland, W. E., Taqqu, M. S., Willinger, W., and Wilson, D. V. On the Self-Similar Nature of Ethernet Traffic. In *Proc. ACM/SIGCOMM'93* (San Francisco, 1993).

[41] Park, K., Kim, G., and Crovella, M. On the Effect of Traffic Self-similarity on Network Performance. In *Proc. SPIE Int'I. Conf. Perf. and Control of Network Sys.* (1997).

[42] Figueiredo, D. R., Liu, B., Misra, V., and Towsley, D. The Autocorrelation Structure of TCP Traffic. Relatório técnico, University of Massachusetts, novembro de 2000.

[43] Feldmann, A., Gilbert, A. C., Willinger, W., and Kurtz, T. G. The Changing Nature of Network Traffic: Scaling Phenomena. *Computer Communication Review 28*, 2 (abril de 1998).

[44] Feldmann, A., Gilbert, A. C., Huang, P., and Willinger, W. Dynamics of IP traffic: A study of the Role of Variability and the Impact of Control. In *Proc. ACM/SIGCOMM'99* (Massachusetts, 1999).

[45] Veres, A., Kenesi, Z., Molnár, S., and Vattay, G. On the Propagation of Long-Range Dependence in the Internet. In *Proc. ACM/SIGCOMM'00* (Stockholm, 2000).

[46] Cao, J., Cleveland, W. S., Lin, D., , and Sun, D. X. Internet Traffic Tends Toward Poisson and Independent as the Load Increase. In *Nonlinear Estimation and Classification* (New York, 2002).

[47] Babic, G., Vandalore, B., and Jain, R. Analysis and Modeling of Traffic in Modern Data Communication Networks. Relatório técnico, Ohio State University, fevereiro de 1998.

[48] Park, K., and Willinger, W. *Self-Similar Network Traffic and Performance Evaluation.* John Wiley and Sons Inc., 2000.

[49] Adler, R. *A Practical Guide to Heavy Tails: Statistical Techniques and Applications.* Birkhauser, 1998.

[50] Veitch, D., and Abry, P. A Wavelet Based Joint Estimator of the Parameters of Long-Range Dependence. In *IEEE Trans. on Info. Theory, Special Issue on Multiscale Statistical Signal Analysis and its applications* (abril de 1999).

[51] Matlab code for the estimation of Scaling Exponents. `http://www.emulab.ee.mu.oz.au/~darryl/secondorder_code.html`.

[52] Cardoso, K. V., de Vasconcellos, S. V., de Rezende, J. F., and Fonseca, N. L. S. Avaliando Estratégias de Push-out no Descarte Seletivo de Pacotes de Tráfego TCP de Curta e Longa Duração. In *Sociedade Brasileira de Redes de Computadores (SBRC'2001)* (maio de 2001).

[53] Cardoso, K. V., de Rezende, J. F., and Fonseca, N. L. S. On the Effectiveness of Push-out Mechanisms for the Discard of TCP Packets. In *IEEE International Conference on Communications (ICC'2002)* (abril de 2002).

[54] Jain, R. *The Art of Computer Systems Performance Analysis.* John Wiley and Sons Inc., 1991.

[55] Floyd, S., and Jacobson, V. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking 1*, 4 (agosto de 1993), 397–413.

[56] Firoiu, V., and Borden, M. A Study of Active Queue Management for Congestion Control. In *infocom00* (março de 2000).

[57] Zhang, Y., and Qiu, L. Understanding the end-to-end Perfomance Impact of RED in a Heterogeneous Environment. Relatório técnico, Cornell University, julho de 2000.

[58] Bonald, T., May, M., and Bolot, J.-C. Analytic Evaluation of RED Performance. In *infocom00* (março de 2000).

[59] May, M., Bolot, J., Diot, C., , and Lyles, B. Reasons Not to Deploy RED. In *In Proc. of 7th. International Workshop on Quality of Service (IWQoS'99)* (junho de 1999), pp. 260–262.

[60] Feng, W., Kandlur, D., Saha, D., and Shin, K. G. BLUE: A new class of active queue management algorithms. Relatório técnico, U. Michigan, abril de 1999.

[61] Ott, T. J., Lakshman, T. V., and Wong, L. H. SRED: Stabilized RED. In *Proceedings of INFOCOM* (1999), vol. 3, pp. 1346–1355.

[62] Arpaci, M., and Copeland, J. A. An Adaptive Queue Management Method for Congestion Avoidance in TCP/IP Networks. In *IEEE/Globecomm'00* (San Francisco, novembro de 2000).

[63] chang Feng, W., Kandlur, D. D., Saha, D., and Shin, K. G. A Self-Configuring RED Gateway. In *Proceedings of INFOCOM 99* (1999), vol. 3, pp. 1320–1328.

[64] Floyd, S., Gummadi, R., and Shenker, S. Adaptive RED: An Algorithm for Increasing the Robustness of RED. Relatório técnico, ACIRI, agosto de 2001.

[65] J. Aweya, M. Ouellette, D. Y. M., and Chapman, A. A control theoretic approach to active queue management. *Computer Networks 36* (2001).

[66] Wydrowski, B., and Zukerman, M. GREEN: An Active Queue Management Algorithm for a Self Managed Internet. In *Proceedings of ICC 2002* (abril de 2002).

[67] Athuraliya, S., Li, V. H., Low, S. H., and Yin, Q. REM: Active queue management. *IEEE Network 15*, 3 (maio de/junho de 2001), 48 – 53.

[68] Hollot, C., Misra, V., Towsley, D., and Gong, W. On designing improved controllers for aqm routers supporting tcp flows. In *Proceedings of IEEE Infocom 2001* (abril de 2001).

[69] Kunniyur, S., and Srikant, R. Analysis and design of an adaptive virtual queue (avq) algorithm for active queue management. In *Proceedings of SIGCOMM 2001* (agosto de 2001).

[70] Recommendation on using the **gentle_** variant of RED. `http://www.aciri.org/floyd/red/gentle.html`.

[71] RED: Discussions of Setting Parameters. `http://www.aciri.org/floyd/REDparameters.txt`.

[72] Christiansen, M., Jeffay, K., Ott, D., and Smith, F. D. Tunning RED for Web Traffic. In *Proc. ACM/SIGCOMM'00* (Stockholm, 2000).

[73] Iannaccone, G., Brandauer, C., Ziegler, T., Diot, C., Fdida, S., and May, M. Comparison of Tail Drop and Active Queue Management Performance for bulk-data and Web-like Internet Traffic. In *Proceedings of IEEE ISCC 2001* (2001).

[74] Blake, S., Black, D. L., Carlson, M., Davies, E., Wang, Z., and Weiss, W. An Architecture for Differentiated Services. *RFC 2475* (dezembro de 1998).

[75] Heinanen, J., Baker, F., Weiss, W., and Wroclawski, J. Assured Forwarding PHB Group. *RFC 2597* (junho de 1999).

[76] Goyal, M., Durresi, A., Misra, P., Liu, C., and Jain, R. Effect of Number of Drop Precedences in Assured Forwarding. In *IEEE/Globecomm'99* (Rio de Janeiro, dezembro de 1999).

[77] Weighted Random Early Detection. `http://www.cisco.com/warp/public/732/Tech/red/`.

[78] Chao, H., Cheng, H., Jenq, Y.-R., and Jeong, D. Design of a Generalized Priority Queue Manager for ATM Switches. *IEEE Journal on Selected Areas in Communications 15*, 5 (junho de 1997), 867–880.

[79] Suter, B., Lakshman, T. V., Stiliadis, D., and Choudhury, A. K. Buffer Management Schemes for Supporting TCP in Gigabit Routers with Perflow Queueing. *IEEE Journal on Selected Areas in Communications* (setembro de 1999).

[80] Floyd, S., and Romanov, A. Dynamics of TCP Traffic over ATM Networks. *IEEE Journal on Selected Areas in Communications 13*, 4 (1995), 633–641.

[81] Goyal, R., Jain, R., Kalyanaraman, S., and Fahmy, S. Improving Performance of TCP over ATM-UBR Service. In *Proc. of ICC'97* (junho de 1997), pp. 1042–1048.

[82] Turner, J. Maintaining High Throughput during Overload in ATM Switches. In *Proc. of INFOCOM'96* (abril de 1996), pp. 287–295.

[83] Lakshman, T., Neidhardt, A., and Ott, T. The Drop from Front Strategy in TCP over ATM. In *Proc. of INFOCOM'96* (abril de 1996), pp. 1242–1250.

[84] Labrador, M., and Banerjee, S. Packet Dropping Policies for ATM and IP Networks. *IEEE Communications Surveys 2*, 3 (1999). http://www.comsoc.org/pubs/surveys.

[85] Clark, D. D., and Fang, W. Explicit Allocation of Best Effort Packet Delivery Service. *IEEE/ACM Transactions on Networking 6*, 4 (agosto de 1998), 362–373.

[86] Cidon, I., Guérin, R., and Khamisy, A. On Protective Buffer Policies. *IEEE/ACM Transactions on Networking 2*, 3 (junho de 1994), 240–246.

Apêndice A

Gerador de Tráfego

O GERADOR de tráfego construído a partir do modelo de agregado proposto foi implementado e testado sobre o simulador NS-2. A seguir é apresentado o código do gerador de tráfego em linguagem OTcl. A versão mostrada é uma simplificação da utilizada no texto desse trabalho. Os recursos removidos dizem respeito à depuração do modelo e ao uso de classes de serviço. Ou seja, toda funcionalidade do modelo de agregado é mantida.

Posterior ao código do gerador de tráfego, é apresentado um exemplo da utilização do mesmo. No exemplo constam as configurações de distribuições do tamanho de transferência utilizadas na obtenção dos resultados que foram apresentados nesse trabalho. Segue ainda o trecho de código em C++ que necessita ser substituído no NS-2 para o correto funcionamento da distribuição Lognormal.

Todo o código descrito a seguir pode ser obtido em versão digital no seguinte URL: **http://www.gta.ufrj.br/~kleber/tfg**.

Arquivo: **tfg.tcl**

```
#
# Copyright (c) GTA/UFRJ 2002. All rights reserved.
#
# License is granted to copy, to use, and to make and to use derivative
# works for research and evaluation purposes, provided that GTA/UFRJ is
# acknowledged in all documentation pertaining to any such copy or
# derivative work. GTA/UFRJ grants no other licenses expressed or
# implied.
#
```

```
# GTA/UFRJ MAKES NO REPRESENTATIONS CONCERNING EITHER THE
# MERCHANTABILITY OF THIS SOFTWARE OR THE SUITABILITY OF THIS SOFTWARE
# FOR ANY PARTICULAR PURPOSE.  The software is provided "as is" without
# express or implied warranty of any kind.
#
# These notices must be retained in any copies of any part of this
# software.
#
# Contributed by Rezende, J.F., http://www.gta.ufrj.br/~rezende
# Designed by Cardoso, K.V., http://www.gta.ufrj.br/~kleber
#
# The procedures in this file deal specially with traffic generation.

Class TrafficGen

TrafficGen instproc init {ns inode enode bbw} {
    $self instvar ns_ inode_ enode_ bbw_ rng_
    $self instvar MAXINT

    set ns_ $ns
    set inode_ $inode
    set enode_ $enode
    set bbw_ $bbw
    set MAXINT 2147483648.0
    set rng_ [new RNG]
    $rng_ seed 0
}

TrafficGen instproc start {} {
    global param
    global post
    $self instvar transfer_count_ ;# transfer count
    $self instvar client_count_   ;# number of created clients
    $self instvar client_onduty_  ;# number of not yet finished clients
    $self instvar ns_ rng_
    $self instvar tracefilefd_ post_ node_

# file used to write events
    set tracefilefd_ [open $param(dir)/temp/trace.tr w] ;

    # set TCP agents parameters
    Agent/TCP set window_ 1000 ;# disable flow control
    Agent/TCP set tcpTick_ 0.1 ;# timer tick = 100ms
    Agent/TCP set packetSize_ $param(tcppsize)
    Agent/TCP set ecn_ 0
    Agent/TCP set useHeaders_ false;
    Agent/TCP set syn_ false;

    set transfer_count_ 0      ;# transfer count
    set client_count_ 0        ;# number of created clients

    puts stderr "rho  => $param(rho)"
    puts stderr "distribution => $param(dist)"

    # launch traffic generation
    $self schedule_initial_traffic
```

```
    # create a post process environment
    set post [new PostProcess/gnuplot $tracefilefd_ $param(dir)/temp/trace.tr]
}

TrafficGen instproc schedule_initial_traffic {} {
    $self instvar ns_ rng_
    $self instvar idle_clients_       ;# array of idle clients
    $self instvar client_onduty_
    $self instvar finished_           ;# number of finished clients

    set idle_clients_ ""
    set client_onduty_ 0;   # number of active clients
    set finished_ 0;        # number of finished transfers

    # flows start randomly between 0 and 1 secs
    set startt [$rng_ uniform 0 1]
    $ns_ at $startt "$self schedule_continuing_traffic 0"
}

TrafficGen instproc schedule_continuing_traffic {delay} {
    global param
    $self instvar MAXINT
    $self instvar ns_ rng_ bbw_

    switch $param(dist) {
mixed {
    set part_ [ns-random]
    set chosen_part_ [expr round ([expr $part_/$MAXINT*100])]
    if {$chosen_part_ <= $param(body_dist_)} {
            set model [new RandomVariable/LogNormal]
            $model use-rng $rng_
            $model set avg_ $param(avg_len_b_)
            $model set std_ $param(std_dev_)
            set L [expr round ([$model value])]
            while {$L == 0} {
               set L [expr round ([$model value])]
}
delete $model
    } else {
            set model [new RandomVariable/Pareto]
            $model use-rng $rng_
            $model set avg_ $param(avg_len_t_)
            $model set shape_ $param(shape_)
            set L [expr round ([$model value])]
            while {$L == 0} {
               set L [expr round ([$model value])]
            }
delete $model
    }
}
fixed {
    set L $param(avg_len)
}
expo {
    set model [new RandomVariable/Exponential]
```

```
    $model use-rng $rng_
    $model set avg_ $param(avg_len_b_)
    set L [expr round ([$model value])]
    while {$L == 0} {
set L [expr round ([$model value])]
    }
    delete $model
}
lognormal {
    set model [new RandomVariable/LogNormal]
    $model use-rng $rng_
    $model set avg_ $param(avg_len_b_)
    $model set std_ $param(std_dev_)
    set L [expr round ([$model value])]
    while {$L == 0} {
           set L [expr round ([$model value])]
    }
    delete $model
}
pareto {
    set model [new RandomVariable/Pareto]
    $model use-rng $rng_
    $model set avg_ $param(avg_len_b_)
    $model set shape_ $param(shape_)
    set L [expr round ([$model value])]
    while {$L == 0} {
           set L [expr round ([$model value])]
    }
    delete $model
}
    }
    # start a new transfer
    $self start_a_client $delay $L
    # time between the transfers
    set inter_delay [expr (1.0/($param(rho)*$bbw_/(1.0*$L)/8.0))]
    # schedule the next transfer
    set next [expr [$ns_ now]+$inter_delay]
    $ns_ at $next "$self schedule_continuing_traffic $inter_delay"
}

TrafficGen instproc start_a_client {inter_delay L} {
    global param
    $self instvar idle_clients_ sources_ app_ tracefilefd_
    $self instvar ns_ rng_
    $self instvar transfer_count_ client_onduty_

    set now [$ns_ now]

    # check for available clients
    set x [llength $idle_clients_]
    if {$x < 5} {
        set i [$self create_a_client]
    } else {
        set i [lindex $idle_clients_ 0]
        set idle_clients_ [lrange $idle_clients_ 1 end]
    }
```

```
    # reset the connection
    $sources_($i) reset
    [$sources_($i) set dst_agent_] reset

    incr transfer_count_
    $sources_($i) set transfer_count_ $transfer_count_
    $sources_($i) set fid_ 1
    $sources_($i) set startt_ $now
    $sources_($i) set inter_delay_ $inter_delay

    # start traffic for that client
    $sources_($i) set transfer_size_ $L
    set len $L
    $app_($i) send $len

    set now [$ns_ now]

    # write a trace
    puts $tracefilefd_ "t: [format %.8f $now ] sc: $i tid: $transfer_count_"

    # increment number of unfinished clients
    incr client_onduty_
}

TrafficGen instproc finish_a_client {clid} {
    global param
    $self instvar ns_ bbw_
    $self instvar client_onduty_
    $self instvar idle_clients_ sources_
    $self instvar tracefilefd_ finished_

    # decrement number of unfinished clients
    incr client_onduty_ -1

    set now [$ns_ now]

    # transfer duration
    set delta [expr $now - [$sources_($clid) set startt_]]

    puts $tracefilefd_ "t: [format %.8f $now] fc: $clid tid: [$sources_($clid)\
set transfer_count_] len: [$sources_($clid) set transfer_size_]\
dur: [format %.8f $delta] act: $client_onduty_"

    # add specific information bandwidth
    set bw [expr [$sources_($clid) set transfer_size_]*8.0 /$delta]
    if { $bw >= $bbw_ } {
        puts $tracefilefd_  "ERROR: Wrong throughtput $bw"
    }
    # normalized (in percentage of the bottleneck bw) bandwidth
    set bwn [expr $bw*100.0/$bbw_]

    puts $tracefilefd_ "t: [format %.8f $now] bw: [format %.8f $bwn]\
rtt: [expr ([$sources_($clid) set srtt_] >> 3) * 0.1] to: [$sources_($clid)\
set nrexmit_]"

    # insert this client in the idle_clients list
```

```
    lappend idle_clients_ $clid
    incr finished_

    set now [$ns_ now]
}

# create a tcp source/dst connection with an attached ftp application
TrafficGen instproc create_a_client {} {
    global param
    $self instvar sources_ app_
    $self instvar ns_ client_count_
    $self instvar tracefilefd_ inode_ enode_

    set now [$ns_ now]

    set client_id [incr client_count_]
    set clients [$ns_ create-connection-list $param(tcp_flavor) $inode_\
TCPSink $enode_ 0]
    set sources_($client_id) [lindex $clients 0]
    set dst_agent [lindex $clients 1]
    $sources_($client_id) set dst_agent_ $dst_agent

    # set up a callback when this client ends.
    $sources_($client_id) proc done {} "$self finish_a_client $client_id"

    set app_($client_id) [new Application/FTP]
    $app_($client_id) attach-agent $sources_($client_id)

    return $client_id
}
```

Arquivo: **post.tcl**

```
#
# Copyright (c) GTA/UFRJ 2002. All rights reserved.
#
# License is granted to copy, to use, and to make and to use derivative
# works for research and evaluation purposes, provided that GTA/UFRJ is
# acknowledged in all documentation pertaining to any such copy or
# derivative work. GTA/UFRJ grants no other licenses expressed or
# implied.
#
# GTA/UFRJ MAKES NO REPRESENTATIONS CONCERNING EITHER THE
# MERCHANTABILITY OF THIS SOFTWARE OR THE SUITABILITY OF THIS SOFTWARE
# FOR ANY PARTICULAR PURPOSE.  The software is provided "as is" without
# express or implied warranty of any kind.
#
# These notices must be retained in any copies of any part of this
# software.
#
# Contributed by Rezende, J.F., http://www.gta.ufrj.br/~rezende
# Designed by Cardoso, K.V., http://www.gta.ufrj.br/~kleber
#
# The procedures in this file deal specially with traffic generation.
```

```
Class PostProcess

PostProcess instproc init { fd nametr } {
    $self instvar nametracefile_
    $self instvar tfilefd_ format_

    set tfilefd_ $fd
    set nametracefile_ $nametr
    set name [$self info class]
    set lname [split $name /]
    set format_ [lindex $lname 1]

    puts "format according $format_ "
}

PostProcess instproc plot_results { bandwidth } {
    $self instvar tfilefd_ nametracefile_
    global param

    puts "\nStart post process"
    puts "Process file $nametracefile_"

    close $tfilefd_

    # open graph file
    set resultsfd [open $param(dir)/results.dat a]

    $self getresults $bandwidth $resultsfd
    close $resultsfd
}

Class PostProcess/gnuplot -superclass PostProcess

# plot rho vs. per-flow rates
PostProcess/gnuplot instproc getresults {bandwidth resultsfd} {
    $self instvar nametracefile_
    global param

    exec awk [$self results] rho=$param(rho) bandwidth=$bandwidth\
warmup=$param(warmup) $nametracefile_ >@ $resultsfd
}

#---------------------------------------------------------------
#                        AWK procedure
#---------------------------------------------------------------
# t: 3.60908622 sc: 1 tid: 463
# $1: time
# $2: <time>
# $3: start a client
# $4: <client id>
# $5: transfer id
# $6: <transfer id>

# t: 3.61228622 fc: 6 tid: 462 len: 5000 dur: 0.00986667 act: 1
# $1: time
```

```
# $2: <time>
# $3: finish a client
# $4: <client id>
# $5: transfer length (bytes)
# $6: <transfer size>
# $7: duration
# $8: <duration>
# $9: active clients
# $10: <active clients>

# t: 3.60662222 bw: 36.80078508 rtt: 0.10000000000000001 to: 0
# $1: time
# $2: <time>
# $3: bandwidth
# $4: <bandwidth>
# $5: rtt
# $6: <rtt>
# $7: timeouts
# $8: <timeouts>

PostProcess/gnuplot instproc results { } {
set awkCodeAll {
    BEGIN {
        sum_active = 0;
        samples_active = 0;
        sum_bw = 0.0;
        samples_bw = 0;
        sum_rtt = 0.0;
        samples_rtt = 0;
        sum_to = 0.0;
        samples_to = 0;
    }
    {
        if ($2 >= warmup) {
            if ($3 == "fc:") {
                sum_active += $12;
                samples_active += 1;
            }
            if ($3 == "bw:") {
                sum_bw += $4;
                samples_bw += 1;
                sum_rtt += $6;
                samples_rtt += 1;
                sum_to += $8;
                samples_to += 1;
            }
        }
    }
#-----------------------------------------------------------------------
# OUTPUT format according column number:
#  1: mean bandwidth
#  2: mean rtt
#  3: mean number of timeouts
#  4: mean number of active clients
#-----------------------------------------------------------------------
    END {
```

```
      ma = sum_active/samples_active;
      mbw = sum_bw/samples_bw;
      mrtt = sum_rtt/samples_rtt;
      mto = sum_to/samples_to;

      printf "%f\t%f\t%f\t%f\t%f\n", rho, mbw, mrtt, mto, ma;
    }
}
}
```

Arquivo: **ex.tcl**

```
set param(endsim) 100.0; # simulation end time
set param(warmup) 10.0; # time to avoid transient (warmup time)

source tfg.tcl; # get traffic generator source
source post.tcl; # get post-simulation source

set param(tcp_flavor) TCP/Reno; # TCP flavor
set param(tcppsize)   500; # TCP segment size
set param(dir)        "."; # directory to put trace file
set param(rho) 0.75; # setup load

# It follows six configuration of transfer size
# Only one must be chosen at a time

# FIXED (fixed transfer sizes)
#  set param(dist)       fixed
#  set param(avg_len)    5000; # transfer size

# HTTP-1 (Pareto distribution)
#  set param(dist)       pareto
#  set param(avg_len_b_) 4100
#  set param(shape_)     1.95

# HTTP-2 (Pareto distribution)
#  set param(dist)       pareto
#  set param(avg_len_b_) 4100
#  set param(shape_)     1.35

# HTTP-3 (Lognormal distribution)
  set param(dist)       lognormal
  set param(avg_len_b_) 4827
  set param(std_dev_)   41008

# HTTP-4 (Hibrid distribution)
#  set param(dist)       mixed
#  set param(body_dist_) 93
#  set param(avg_len_b_) 27599.75
#  set param(std_dev_)   59713.65
#  set param(avg_len_t_) 1463000
#  set param(shape_)     1.1

# HTTP-5 (Hibrid distribution)
#  set param(dist)       mixed
```

```
#  set param(body_dist_) 88
#  set param(avg_len_b_) 7247
#  set param(std_dev_)   28765
#  set param(avg_len_t_) 10558.46
#  set param(shape_)     1.383

set ns [new Simulator]; # instantiate the simulator

# create nodes for a very simple topology
set n1 [$ns node]
set n2 [$ns node]
set bbw 10000000.0; # bottleneck link capacity
$ns duplex-link $n1 $n2 $bbw 1ms DropTail; # setup link between n1 and n2

# instantiate the traffic generator
set tfg [new TrafficGen $ns $n1 $n2 $bbw]

$tfg start; # start traffic generation
$ns at $param(endsim) "finish"; # schedule simulation end

# things to do when the simulation ends
proc finish {} {
    global post bbw

    if {[info exists post]} {
# store the following results in the file results.dat:
# rho - mean throughput - mean RTT - number of timeouts -
# - mean number of simultaneous connections
        $post plot_results $bbw
    }
    exit 0
}

$ns run
```

Arquivo: **rng.h** (código do NS-2 que deve ser substituído para o funcionamento adequado da distribuição Lognormal)

```
//      inline double lognormal(double avg, double std) {
//              return (exp (normal(avg, std)));
//      }
double lognormal(double avg, double std) {
        double c, sigma, L;

        sigma = sqrt(log(pow(std,2)/pow(avg,2)+1));
        c = log(avg)-(0.5*log(pow(std,2)/pow(avg,2)+1));
        L = exp(c + normal(0,1)*sigma);

        return L;
}
```