UNIVERSIDADE SALVADOR – UNIFACS
DEPARTAMENTO DE ENGENHARIA E ARQUITETURA
CURSO DE ENGENHARIA ELÉTRICA

MARCOS PORTNOI

ESTUDO DE CARACTERÍSTICAS DE FONTES DE TRÁFEGO PARA REDES DE COMPUTADORES MULTI-SERVIÇO

Salvador – BA
2003

MARCOS PORTNOI

ESTUDO DE CARACTERÍSTICAS DE FONTES DE TRÁFEGO PARA REDES DE COMPUTADORES MULTI-SERVIÇO

Monografia apresentada ao Curso de graduação em Engenharia Elétrica, Universidade Salvador – UNIFACS, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Sergio de Figueiredo Brito

Salvador – BA

2003

AGRADECIMENTOS

Ao Núcleo Interdepartamental de Pesquisas em Redes de Computadores (NUPERC) da UNIFACS, pelo apoio e oportunidade dada nos anos em que fui pesquisador em Iniciação Científica.

Aos meus professores André Valente, Armando, Adelmo, Leonardo Teixeira, Luciene, Paulo Menezes (outro admirador de Carl Sagan), Milton Sampaio, Paulo Vitor, Maria das Graças, Paulo Pedreira, Marcelo Rios, Maria Olívia, Ricardo Salgueiro, Marise Mota, Maria Ângela, Maria das Graças Almeida, Celso Saibel.

Professor José Carlos, com sua visão prática e mente rápida, presenteou-me com o sentimento de orgulho de ser engenheiro.

Professor Kleber Freire, sempre disponível a me auxiliar nas minhas constantes buscas por auxílio.

Professor Sérgio Bello, um incentivador incansável.

Professor Paulo Lobo, que desde o começo alimentou meu espírito pesquisador.

Professora Mônica Aguiar, sempre atenciosa.

Minha colega de escola e professora Mariana Torres Strauch. Como dizíamos em épocas dantes... Jacobina.

Professor Euclério Ornellas, um verdadeiro formador, devorador de livros, querido unanimamente.

Professora Alzineide Dantas, por sua seriedade e por ter redespertado meu gosto por vocabulário requintado e nobre.

Professor William Giozza, preciso, conhecimento enciclopédico.

Professor Joberto Martins, pessoa de conhecimento vasto e impecável, e acima de tudo, de trato e gentileza inabaláveis. Um autêntico *gentleman*. Uma honra ter podido ser seu aluno.

Professor Sergio de Figueiredo Brito. Professor, orientador, estimulador, incentivador, auxiliador, confidente, mentor, mestre e amigo. Seu empenho e sua confiança depositados em mim são muito mais do que eu jamais poderia pedir.

“When it’s my moment in the sun
Oh, how beautiful I’ll be
But in a normal sort of way
Like I am you and you are me”

-- Eef Barzelay (Clem Snide Band)

RESUMO

Esta monografia tem como objetivo o estudo de quatro tipos de fontes geradoras de tráfego para redes de computadores, com vistas a sua caracterização. Expõe-se modelos descritivos das fontes e, a partir de códigos disponíveis na Internet, inclui-se no escopo deste trabalho programas cujos algoritmos descrevem o processo de geração dos tráfegos.

Palavras-chave: fonte, tráfego, gerador, rede de computadores, voip, self-similar, fractal, vbr, mpeg, pareto, poisson, lan, ethernet.

ABSTRACT

This work intends to study four types of traffic sources for computer networks, aiming at the characterization of such sources. Descriptive models of the sources are presented and, based on code available in the internet, this work includes algorithms that describe the process of generating the traffic.

Keywords: source, network traffic, generator, computer network, voip, self-similar, fractal, vbr, mpeg, pareto, poisson, lan, ethernet.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ATM	Asynchronous Transfer Mode
B-ISDN	Broadband Integrated Services Digital Network
CBR	Constant Bit Rate
DCT	Discrete Cosin Transform
DTMF	Dual Tone Multi Frequency
F-ARIMA	Fractional Autoregressive Integrated Moving Average Process
FDM	Frequency Division Multiplexing
fFGN	Fast Fractional Gaussian Noise
FFT	Fast Fourier Transform
HDTV	High Definition Television
ISDN	Integrated Services Digital Network
ITU	International Telecommunications Union
ITU-T	Telecommunication Standardization Sector of the International Telecommunications Union
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
LRD	Long Range Dependence
MCU	Multipoint Control Units
MPEG	Moving Picture Experts Group
OTcl	Object Tool Command Language
PABX	Private Automatic Branch eXchange
PAL	Phase Alternation Line
PBX	Private Branch eXchange
PCM	Pulse Code Modulation
PDU	Protocol Data Unit
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RDSI	Rede Digital de Serviços Integrados
RMD	Random Midpoint Displacement
RTCP	Real Time Control Protocol
RTP	Real-Time Transport Protocol

SECAM	Système Electronique Couleur Avec Mémoire
SRD	Short Range Dependence
TDM	Time Division Multiplexing
WAN	Wide Area Network
WWW	World Wide Web

SUMÁRIO

1. INTRODUÇÃO

A engenharia freqüentemente é chamada a propor melhorias para processos vigentes ou para apresentar soluções para problemas. A Avaliação de Desempenho de Sistemas oferece técnicas para abordagem; uma delas consiste em **modelar** um determinado **sistema**, contendo as variáveis de interesse para análise, de forma que este modelo possa ser tratado matematicamente. O tratamento do modelo resultará em diversos resultados de acordo com as situações desejadas, visualizados através das variáveis de interesse. Num sistema de redes de computadores, estas variáveis podem ser: o atraso fim-a-fim, pacotes perdidos, vazão, dentre outras [PORTNOI, 2002].

Uma vez proposto de forma matemática, o modelo pode ser resolvido através de métodos analíticos ou simulação [JAIN, 1991]. O método analítico, ou algébrico, requer profundo conhecimento matemático, mas confere exatidão nas respostas. A resolução, dependendo do modelo matemático, pode tornar-se extenuante, o que obriga a simplificações do modelo. Estas simplificações podem culminar em imperfeições na representação do modelo. O método por simulação, por outro lado, permite a confecção de modelos complexos e a resolução destes com menor desenvolvimento matemático. Emprega-se, aqui, poder computacional para as iterações numéricas. A depender dos resultados desejados, este poder computacional pode ser significativo.

O ponto importante é que, para confecção de um modelo eficiente para ser tratado pelo simulador, é preciso estudar o fenômeno ou sistema de modo a bem compreendê-lo. Aqui, usa-se das ferramentas da Estatística ou Processos Estocásticos, de forma a produzir as equações matemáticas.

Em redes de computadores, um dos campos de estudo é a caracterização e modelagem de fontes de tráfego. Pode-se entender como "fonte de tráfego", tipicamente, uma aplicação de redes de computadores gerando um conjunto de bits, que por sua vez trafegam pela rede em um dado período. A fonte de tráfego terá características próprias. Neste texto, será possível verificar que, por exemplo, uma fonte do tipo Voz sobre IP alterna períodos de atividade e períodos de dormência; vídeo digital comprimido, por sua vez, exibe característica de taxa de bits variável ao longo do tempo.

Uma alternativa à modelagem estatística/matemática das fontes de tráfego em redes de computadores consiste em "gravar" o tráfego real, através de equipamentos especiais como os analisadores de redes ou "*sniffers*", e então reproduzir a gravação (que recebe o nome de *tracing*). Os modelos de interesse podem ser simulados através desta reprodução, que espelhará exatamente o comportamento da fonte original.

A caracterização eficiente de fontes de tráfego (resultando em bons modelos) é importante para áreas relacionadas à gerência de desempenho de redes de serviços integrados, como por exemplo, garantir que todos os tipos de aplicação recebam a qualidade de serviço apropriada [SCHWARTZ, 1996]. Os modelos estocásticos dos fluxos de tráfego são amplamente usados na predição de desempenho dos sistemas; assim, é essencial descrever cuidadosamente uma fonte de tráfego sob escrutínio de forma a asseverar que os modelos usados de fato levam a resultados úteis de desempenho.

O objetivo deste trabalho é, pois, tomar quatro tipos fontes de tráfego encontradas em redes de computadores multi-serviço: (a) Voz sobre IP; (b) Vídeo Digital MPEG; (c) Auto-Similar; e (c) FTP (*File Transfer Protocol*). O estudo de cada tipo procurará identificar as aplicações responsáveis por sua geração. Então, será feita uma caracterização e apresentação

de uma modelagem para a fonte. Por fim, anexar-se-á um exemplo de código ou programa de computador para geração sintetizada do tráfego, com vistas à simulação da fonte.

O escopo das análises será qualitativo. Evitar-se-á desenvolvimentos matemáticos, preferindo-se uma caracterização prática à densidade algébrica. Os códigos inseridos como exemplo, em sua maioria, não são aptos a funcionar de forma independente, mas sim fazem parte de um ambiente de simulação [NS, 2003]. O intuito é ilustrar algoritmos de geração de tráfego simulado. Foge também ao intuito desta monografia prover comentários detalhados sobre os códigos exemplificados.

A descrição de fontes de tráfego permite um maior conhecimento de seu comportamento, visando a geração adequada do modelo em questão. O gerador de tráfego faz parte do conjunto de um simulador de redes de computadores, a exemplo de [NS, 2003], e que torna possível aferir o desempenho de um determinado projeto de rede sob um certo tipo de tráfego. Como ilustração, imagine-se uma rede corporativa composta de 500 computadores, montada sob Ethernet a 10Mbps e com um *link* para a internet de 256kbps. Deseja-se conhecer o comportamento desta rede se submetida a tráfego de videoconferência externa (via internet) e também interna, e quais os atrasos esperados. Com um simulador de redes, monta-se a topologia e estrutura da rede (que pode ainda nem existir fisicamente) e submete-se este modelo a vários cenários simulados. Para que isso seja possível, torna-se indispensável a figura do gerador de tráfego, que, sob o ponto de vista da teoria de filas, é um gerador de eventos. Com o tráfego gerado corretamente, obtêm-se as estatísticas de interesse.

Este trabalho será desenvolvido sob a seguinte forma. Para cada fonte de tráfego eleita, identificar-se-á inicialmente a aplicação geratriz relacionada. Em seguida, caracterizar-se-á as fontes sob o aspecto comportamental, apresentando um modelo descritivo. E,

finalmente, disponibilizar-se-á um exemplo de código representando um algoritmo para geração do tráfego da fonte analisada.

2. FONTE DE VOZ

A voz, ou fonação, é um som produzido pelo ser humano a partir de seu aparelho fonador. Especificamente, o ar é expelido pelo pulmão e é modulado pelas cordas vocais instaladas na faringe; então, é ressonado pelo peito, garganta, fossas nasais e cavidades na boca e articulado pelo palato, língua, lábios e dentes, de forma que os sons resultantes, combinados em unidades fonéticas – a **fala -** sejam inteligíveis como informação para outro interlocutor humano (ainda que o ser humano use da voz para se comunicar com outras formas de vida).

Os sons são ondas mecânicas que viajam por um fluido, em geral o ar. Essas ondas são compostas por zonas de compressão e descompressão do ar (ou fluido), ou zonas de maior/menor pressão, trafegando no ar a uma velocidade aproximada de 1.000 Km/h, ao nível do mar. O som, na verdade, é a interpretação dessas ondas dada pelo cérebro, quando captadas pelo ouvido. Assim, a voz, como som, tem natureza analógica.

2.1. TELEFONIA E SISTEMAS DE COMUTAÇÃO

Sistema mais importante de transferência de voz à distância, a telefonia nasceu inteiramente baseada em tecnologia analógica. Em essência, é a captação do som em uma ponta, através de um transdutor analógico como o microfone, que converte as impressões sonoras em sinais elétricos. Os sinais elétricos são então transmitidos à outra ponta, onde são novamente convertidos em ondas sonoras por outro transdutor analógico, tipicamente o alto-falante.

2.1.1. Comutação de Circuitos: conceitos

Esse foi o sistema sobre o qual montou-se todo o sistema de telefonia (daí o nome pelo qual o sistema de telefonia é conhecido: PSTN – *Public Switched Telephone Network*, ou

rede pública de telefonia comutada): analógico, baseado em comutação de circuitos, com terminais simples nas pontas e concentração da inteligência nas centrais telefônicas [TELEFONIA IP, 2002].

A comutação de circuitos pode ser entendida como uma reserva de um circuito físico (um "fio") entre a origem e o destino, durante todo o tempo da conexão ou chamada. Uma interpretação mais atual da comutação de circuitos consiste em uma reserva de banda passante, de modo que outras aplicações não possam compartilhar esta banda (ou circuito) enquanto estiver reservada. Assim, haverá um tamanho de banda dedicado para uso de determinada aplicação ou usuário, mesmo que o meio físico (cabo metálico ou fibra ótica, por exemplo) esteja transportando vários "circuitos" concomitantemente [GUIMARÃES, 1999]. Estes circuitos podem ser denominados *circuitos virtuais*.

O transporte de vários "circuitos" num mesmo meio físico é possibilitado através do mecanismo da *multiplexação*. Por exemplo, usando a TDM (*Time Division Multiplexing*, ou Multiplexação por Divisão de Tempo), uma "janela" (ou *slot*) de tempo é reservada para cada usuário, de modo que durante esta janela, o usuário utiliza o meio sozinho. O intervalo de tempo de cada *slot* é tipicamente bastante reduzido, o que permite que o usuário tenha acesso a ele várias vezes por segundo e que vários usuários também tenham possibilidade de usar o meio. Usando a FDM (*Frequency Division Multiplexing*, ou Multiplexação por Divisão em Freqüência), a cada usuário é especificada uma freqüência de modulação no espectro, diferente dos demais usuários. Assim, os diversos sinais modulados podem trafegar no mesmo meio e serem demodulados no destino, sem que se misturem. Observar que, no TDM, os usuários alternam-se no tempo, em instantes bem pequenos, mas durante os quais aqueles têm acesso à toda largura de banda do meio. Em FDM, a banda é subdividida no espectro de freqüência em bandas menores, que são então designadas para cada usuário (ou aplicação).

O sistema de telefonia atual em grande parte baseia-se numa mistura de circuitos virtuais com comutação de circuitos físicos. Basicamente, a comutação de circuitos físicos é feita entre os usuários finais e a central telefônica local[1], e os circuitos virtuais são usados na comunicação entre as centrais. Os circuitos virtuais são inteiramente digitais, entretanto, o que obriga a transformação do sinal de voz em um sinal digital, a *digitalização* (que acontece na primeira central telefônica, que faz a ligação com o usuário).

2.1.2. Digitalização – Pulse Code Modulation (PCM)

A digitalização (transformação de um sinal analógico em um sinal digital) é feita em três etapas: amostragem, quantização e codificação [CARVALHO, 2001; LATHI, 1998, TELEFONIA IP, 2002]. Antes destas etapas, aplica-se também a *filtragem*, que destina-se a limitar o espectro de freqüências do sinal a ser digitalizado a uma determinada freqüência máxima ou a uma faixa de freqüência. Deste modo, as freqüências fora do espectro limitado são simplesmente descartadas.

A *amostragem* consiste em dividir o tempo em instantes discretos. Durante cada instante discretizado, a amplitude do sinal é então medida. Essa medição é então traduzida segundo uma escala de níveis também discretizada: a *quantização*. Portanto, as medições são arredondadas para os níveis quantizados mais próximos. Após a amostragem e quantização, o sinal será discreto tanto no tempo, quanto na amplitude.

O próximo processo, a *codificação*, consiste em representar o sinal amostrado e quantizado no formato digital binário, ou seja, usando 0's e 1's. Isso é feito considerando-se que existem L níveis quantizados, finitos, possíveis para as amplitudes do sinal discreto.

[1] Cada telefone tem um par de fios que vão destes até a central telefônica mais próxima, chamada Central Telefônica Local. Esta conexão entre os usuários e as centrais telefônicas locais é conhecida como *local loop* (loop ou enlace local).

Estes níveis são então representados por um número binário. Por exemplo, se existem 8 níveis quantizados, pode-se representar cada nível por 3 bits ($2^3 = 8$). Observar que os níveis em si podem significar qualquer valor de grandeza; o nível 1 pode representar 5 Volts, o nível 2, 7 V e assim por diante. O nível 1 em binário é codificado como 001, o nível 2 como 010, etc.

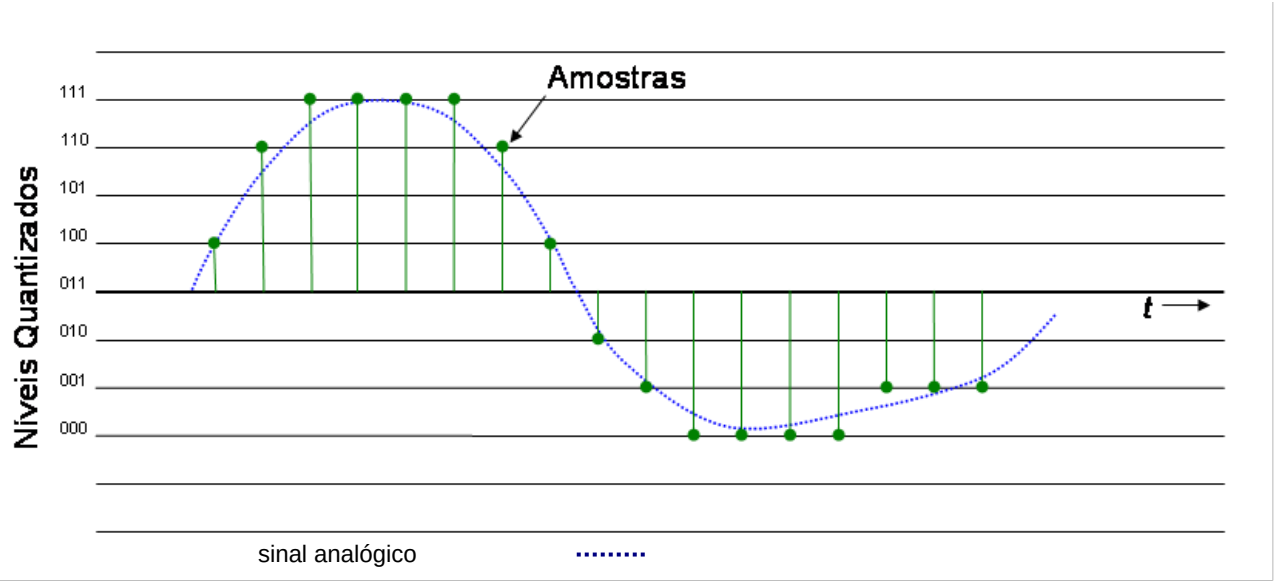


Figura 1: Amostragem, quantização e codificação de um sinal analógico (PCM). Um código binário é designado para cada nível quantizado (000, 001, 010, etc.), que é assumido de acordo com a amplitude medida do sinal original analógico, aproximando-o para o nível quantizado mais próximo.

Todo o processo de digitalização descrito acima caracteriza a Modulação por Código de Pulsos, ou **PCM** (*Pulse Code Modulation*), que é o sistema mais largamente usado dentre todos os sistemas de modulação de pulsos e é o coração do sistema moderno de telefonia. Neste sistema, a inteligibilidade da voz é mais importante que sua fidelidade. Testes subjetivos mostram que uma banda começando em 300 Hz até 3400 Hz é suficiente para transmissão da voz humana sem perda de compreensão. Assim, para telefonia, a padronização especifica um filtro passa-baixa com freqüência de corte de 3400 Hz.

Pelo Teorema de Nyquist [LATHI, 1998; CARVALHO, 2001], a freqüência de amostragem mínima é de 2*B*, onde *B* é a freqüência máxima do sinal amostrado, no caso 3400 Hz. O padrão internacional especifica para telefonia a freqüência de 8 KHz, ou 8000 amostras por segundo. Esta taxa é intencionalmente mantida mais alta que a taxa de Nyquist de 6,8 kHz de modo a evitar filtros não realizáveis (não causais) requeridos na reconstrução do sinal. Cada amostra é então quantizada em 256 níveis. Cada nível requer pois 8 pulsos binários ou 8 bits ($2^8 = 256$) para codificar cada amostra. No receptor, um decodificador interpreta o código recebido e reconstrói uma amostra com tensão igual ao valor médio do intervalo de quantização correspondente àquele código. O conjunto codificador-decodificador é normalmente construído sob um mesmo circuito integrado (ou faz parte de um mesmo programa de computador), denominado ***codec***.

2.1.3. O "Canal" PCM

Na telefonia, portanto, para 8.000 amostras por segundo, sendo que cada amostra tem 8 bits, um total de 8.000 * 8 = 64.000 bits/seg (bps ou *bits per second*) ou pulsos binários por segundo são requeridos para transportar uma conexão de voz. Este tamanho de 64Kbps é conhecido como "canal PCM" e serve de base para vários tamanhos múltiplos de banda passante comerciais utilizados atualmente. Virtualmente todos os intervalos de tempo dentro do sistema telefônico são múltiplos de 125μs (1/8000 amostras por segundo = 125μs/ amostra).

A telefonia digital convencional tende a ser digital apenas na conexão entre centrais telefônicas. O desafio é digitalizar também a chamada "última milha", ou seja, a conexão entre as centrais telefônicas e os usuários, que ainda se utiliza majoritariamente de par de fios de cobre e aparelhos analógicos simples. Em outras palavras, a conversão analógico/digital

(A/D) dos sinais de voz, ao invés de ser feita nas centrais telefônicas, seria feita já na ponta do usuário, transformando todo o sistema em uma rede puramente digital.

Pode-se aqui mencionar a iniciativa da ISDN (*Integrated Services Digital Network*), ou RDSI (Rede Digital de Serviços Integrados) em português. A princípio, o ISDN (nome do serviço oferecido pelas companhias telefônicas) tencionava digitalizar todo o caminho telefônico, desde entre centrais até o usuário, permitindo também agregar outros serviços na linha telefônica, como transmissão de dados e fax, de forma concomitante e independente, muitos anos antes do fenômeno de popularização da internet. Em função do alto custo e desinteresse por parte dos usuários, este serviço não logrou êxito.

Com a explosão da internet, algumas companhias telefônicas, especialmente no Brasil, tentaram ressuscitar o ISDN [TELEMAR, 2001], porém sem sucesso. Sendo uma tecnologia antiga, ela perdeu em chances de competir com formas de acesso à internet em banda larga mais rápidas e baratas, como xDSL, cabo coaxial e rádio. Atualmente, a sigla ISDN (mais especificamente, B-ISDN, de *Broadband Integrated Services Digital Network*) volta a ser usada por autores como [CHANDRA, 1999; HUANG, 1995b; MENDES F^{o}., 1996] para denominar as próprias redes de computadores, maiores e mais possantes, agregando serviços como voz, vídeo sob demanda e dados, entre outros.

2.1.4. Comutação de Pacotes: conceitos

Uma rede de comutação de pacotes, tipicamente redes de dados digitais, não reserva uma banda passante entre a origem e o destino. Esta tecnologia consiste em colher as mensagens ou dados, já previamente divididas em unidades menores por camadas superiores na pilha de protocolos (chamados *pacotes* na tecnologia IP e *células* na tecnologia ATM), de modo que cada pacote trafegue na rede até o destino, podendo tomar rotas diferentes uns dos outros e também compartilhando a rede com pacotes de outros usuários.

Camadas da pilha de protocolos de rede recebem dados da camada superior e agregam-lhe informações de controle, na forma de um cabeçalho, e então remetem o conjunto resultante (denominado PDU – *Protocol Data Unit*, ou unidade de dados do protocolo) para a camada imediatamente abaixo. Do ponto de vista dos dados, este procedimento é transparente. Uma analogia costumeira é o serviço de correios. Uma carta escrita é posta num envelope, que por sua vez recebe a marcação de destino e remetente. O envelope então é colocado num malote, que então é acondicionado em algum outro meio de transporte, como caminhão, aeronave, trem, carteiro, a fim de atingir a central de distribuição de destino. O malote é descarregado da aeronave ou caminhão, e em seguida o malote é esvaziado. O envelope, ao atingir a pessoa de destino, é finalmente aberto, expondo a carta e seu conteúdo. A carta recebeu aqui uma série de "cápsulas" a fim de permiti-la atingir o destino. Um envelope com endereços, então um malote com mais dados de controle, um caminhão ou aeronave com seus próprios mapas de encaminhamento. Para a carta escrita, todo este processo externo é desconhecido e irrevelevante, assim como, em princípio, o conteúdo do envelope é irrelevante ao próprio envelope, ao malote e às transportadoras de malote. O processo executado por uma camada da pilha de protocolos de rede de tomar a PDU de uma camada superior, agregar um cabeçalho e encaminhar o resultado a uma camada inferior é denominado *encapsulamento* [TANENBAUM, 1996].

Cada pacote da rede de comutação contém, pois, um cabeçalho com informações de controle como origem e destino e são depois remontados na seqüência correta no destino (serviço este executado por camadas superiores na pilha de protocolos, tipicamente a camada de transporte).

2.1.5. Comutação de Circuitos x Comutação de Pacotes

É interessante aqui usar uma analogia simplificada para caracterizar as redes de comutação de circuitos e de pacotes, que são as redes de transporte rodoviário e ferroviário. A ferrovia pode ser entendida como uma rede de comutação de circuitos. Há um caminho reservado, onde só um trem pode trafegar a um instante, imperturbável, e para um destino fixo. A rodovia, por sua vez, é compartilhada entre diversas unidades menores (os automóveis), que viajam para destinos diferentes dentre si. Observe-se que a ferrovia é mais imune a atrasos, porém demanda um grande volume de carga para que seja viável (uma vez que um trem só viaja a destinos pré-determinados, obrigando toda sua carga a comparecer aos mesmos destinos). A rodovia permite o tráfego mais flexível de cargas menores e independentes entre si, porém é mais sujeita a congestionamento (= atraso).

É factível então inferir que as redes de comutação de circuitos têm como vantagem a existência de uma banda passante dedicada, que por sua vez pode garantir requisitos de qualidade de serviço (QoS – *Quality of Service*) para alguma aplicação, como por exemplo transmissão de vídeo fluida ou voz sem atrasos. Contudo, o estabelecimento de um circuito dedicado pode significar uma utilização ineficiente dos recursos da rede. Os períodos de não-transmissão (por exemplo, na telefonia, quando os interlocutores estão em silêncio e que são estimados em 60% do tempo total) resultam em desperdício da banda passante, já que esta permanece reservada e não pode ser usada por outras aplicações em transmissão. Algoritmos de compressão buscam justamente eliminar períodos de silêncio ou redundância de informação (ex.: imagens que não mudam ao longo do tempo) de forma a que o meio de transmissão não fique ocioso [GUIMARÃES, 1999].

As redes de comutação de pacotes oferecem como vantagem a utilização mais eficiente dos recursos de rede, pois não há, a princípio, reserva destes recursos[2]. Desta característica advém-lhes a desvantagem para aplicações que têm requisitos mais rigorosos de QoS, como aplicações de tempo real. Como o tráfego de pacotes é variável, podem ocorrer congestionamentos e por conseguinte atrasos, que prejudicam as aplicações que necessitam de tempos de transmissão pequenos e/ou constantes. Outra desvantagem é que os pacotes ou quadros de informação devem conter um cabeçalho de controle, com dados de roteamento e seqüência. Este cabeçalho não se constitui em informação útil, pois não faz parte da mensagem original: assim, parte da banda passante do meio é consumida na transmissão de dados de controle, não relevantes para os usuários.

Paralelamente às grandes redes de telefonia, surgiram as redes de computadores. Inicialmente as redes de longa distância (WAN – *Wide Area Networks*) cresceram utilizando-se da infraestrutura proporcionada pelas empresas de telefonia, coexistindo de maneira independente (e que geraram tecnologias de rede baseadas em circuitos virtuais, como a X.25 e ATM). Mesmo que uma determinada tecnologia de rede compreenda o estabelecimento de circuitos virtuais, as redes de computadores são essencialmente redes de comutação de pacotes (no caso das WAN's). Pela teia de conexão de meios físicos, trafegam dados e aplicações de diversos usuários ao mesmo tempo.

De maneira geral, mesmo atualmente, onde a infra-estrutura digital das empresas de telecomunicações é usada tanto para transportar dados como para voz digitalizada, os dois serviços – telefonia e redes de computadores – são ainda distintos. A telefonia, ou o transporte de voz, pode contudo ser abordada como um serviço oferecido por uma rede de

[2] Há propostas de algoritmos para garantia de QoS que prevêem reserva de recursos, a exemplo de RSVP e IP-IntServ [BRITO, 2001].

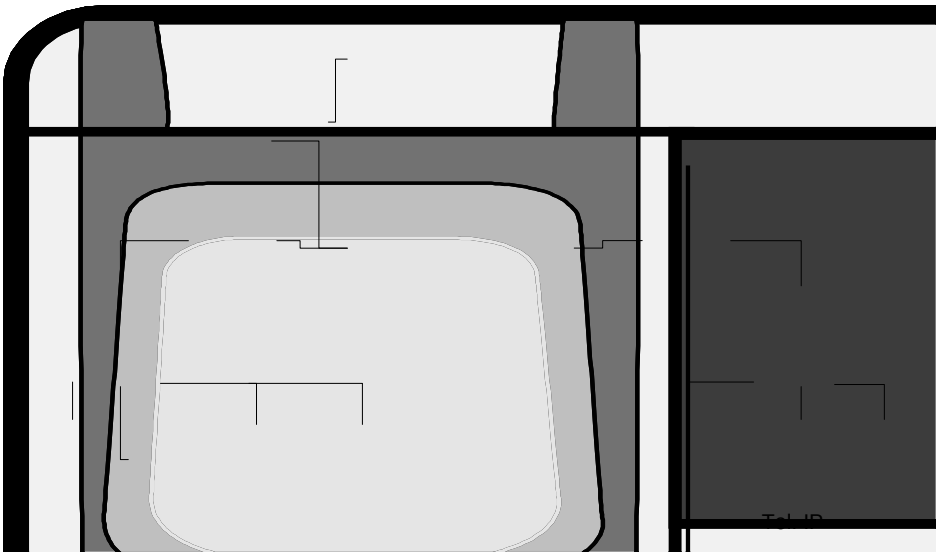
computadores, e não uma aplicação independente. Assim, ter-se-ia, ao invés de aparelhos telefônicos, computadores (dedicados ou não), que seriam responsáveis pela captação da voz, digitalização, encapsulamento desses sinais digitalizados num protocolo adequado de rede e envio através de redes comuns de computadores para o destino, onde outros computadores processariam o caminho inverso de desencapsulamento, conversão digital para analógico e geração da voz.

2.2. VOZ SOBRE IP (VOIP)

A **Voz sobre IP**, ou VoIP, é uma proposta tecnológica para transformar a telefonia em uma aplicação sobre redes de computadores, utilizando-se do popular protocolo IP de redes para transmitir voz digitalizada em tempo real [CHUAH, 2002]. Isso pode ser feito de maneira completa, ou seja, a digitalização da voz e encapsulamento é feita diretamente nos aparelhos de telefone, ou de forma híbrida. No conceito híbrido, usa-se um sistema de PABX[3] (*Private Automatic Branch eXchange*, ou como é conhecido em inglês, PBX, de *Private Branch eXchange*) convencional e integra-se esse PABX a um *gateway* (*switch* ou roteador), que por sua vez fará a conversão para VoIP (Figura 2).

[3] *Private Automatic Branch eXchange*, ou como é conhecido em inglês, PBX, de *Private Branch eXchange.* Sistema telefônico empresarial interno, que comuta usuários dentro da corporação entre si (ramais) e também compartilha um certo número de linhas telefônicas externas (conhecidas como “troncos”) entre estes usuários.

(a)



(b)

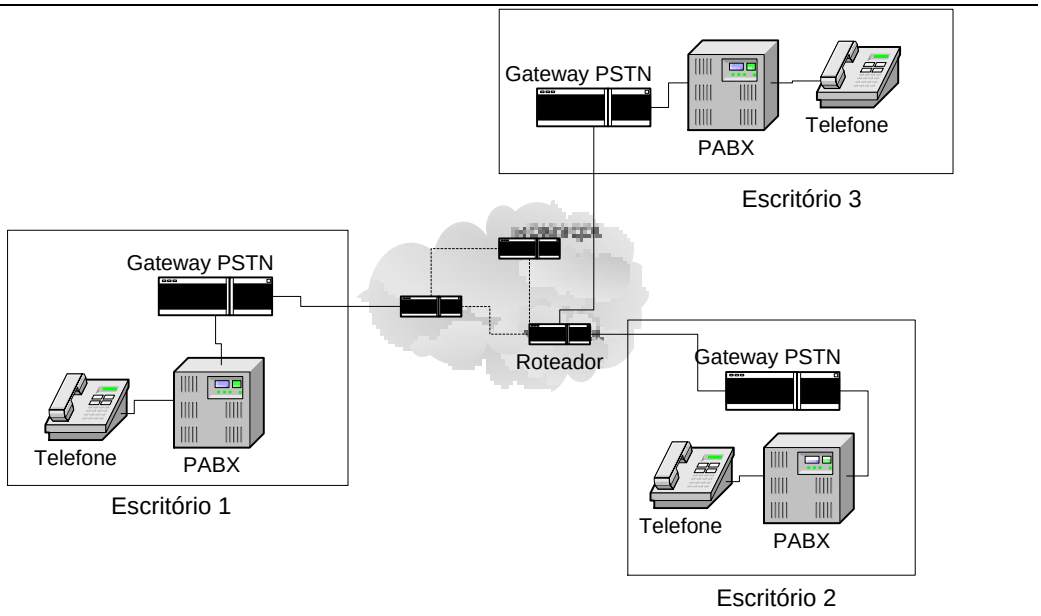


Figura 2: (a) VoIP fim-a-fim. A figura ainda mostra a possibilidade de conexão com a PSTN, através de um *gateway*. (b) VoIP híbrido, com centrais PABX e telefones convencionais nas pontas. O *gateway* encarrega-se de converter telefonia convencional em VoIP.

Pode-se assim afirmar que o dispositivo para VoIP, seja o telefone específico ou *gateway*, incumbe-se de dois procedimentos básicos: executa o *codec* para codificação ou decodificação da voz para digital ou digital para voz, e o encapsulamento-desencapsulamento dos sinais digitais em pacotes IP ou vice-versa. Existem diversos *codecs* para digitalização de voz [GOODE, 2002], alguns relacionados na Tabela 1. O *codec* ITU (*International*

Telecommunications Union) G.711 é o padrão base para o PSTN, e de uso universal. Numa rede de comutação de pacotes, vários *codecs* podem ser utilizados, geralmente negociados pelos dispositivos de comunicação dentre os suportados por eles.

Tabela 1: Características de codecs de voz (G.xxx são definidos pelo ITU; IS-xxx são definidos pela TIA). [GOODE, 2002]

Codec	Algoritmo	Tamanho do Quadro	Taxa Normal
G.711	PCM	125 µs	64 Kbps
G.722	PCM	125 µs	48, 56 ou 64Kbps
G.726	ADPCM	125 µs	32Kbps
G.728	LD-CELP	625 µs	16Kbps
G.729(A)	CS-ACELP	10 ms	8 Kbps
G.729e	Hybrid CELP	10 ms	11,8 Kbps
G.723.1(6.3)	MPC.MLQ	30 ms	6,3 Kbps
G.723.1(5.3)	ACELP	30 ms	5,3 Kbps
IS-127	RCELP	20 ms	Variável, 4,2Kbps média
AMR	ACELP	20 ms	Variável, 4,75-12,2 Kbps

Existem também protocolos de aplicação específicos para operação com multimídia sobre IP, responsáveis pelo encapsulamento, compressão e controle de erros e atraso. Para VoIP, tem-se o padrão H.323 do ITU, que é um conjunto de recomendações e pilha de protocolos para comunicações multimídia em tempo real através de pacotes de rede, como o IP.

Para VoIP, o H.323 identifica as seguintes entidades: terminais, *gateways*, *gatekeepers* e MCU's (*Multipoint Control Units*, ou unidades de controle multiponto). Vejamos resumidamente a função de cada entidade.

- Os terminais H.323 são os pontos de comunicação em tempo real (entre si). Suportam voz, vídeo e dados. Podem ser telefones IP ou aplicações rodando em computadores.

- O *gateway* faz conversão de protocolos e transcodificação entre pontos H.323 e pontos não-H.323. Por exemplo, um *gateway* VoIP faz a tradução de formatos de transmissão e procedimentos de sinalização entre uma rede de comutação de circuitos de telefonia e uma rede de comutação de pacotes. Pode ainda realizar a codificação e compressão de voz e é capaz de gerar e detectar sinais DTMF (os sinais de tom da telefonia convencional).
- O *gatekeeper* H.323 faz o controle de admissão e tradução de endereços. Vários *gatekeepers* podem manter comunicação entre si de modo a coordenar essas tarefas. Redes com *gateways* VoIP podem opcionalmente incluir os *gatekeepers* para traduzir números de telefone (endereços E.164) para endereços de transporte (endereço IP e porta). O *gatekeeper* pode ser um equipamento específico, ou integrar outros elementos da rede, como os terminais ou os *gateways*. Outras funções do *gatekeeper*, quando presente na rede, são: controle de largura de banda (quanta banda cada terminal pode usar); controle de sinalização de chamadas; serviço de diretório; manutenção de relatórios de chamadas (duração, etc.).
- Os MCU's administram conferências entre três ou mais terminais H.323, que devem então se conectar a ele para isso.

O padrão H.323 engloba os seguintes protocolos:

- *codec* áudio – faz a codificação e decodificação de sinais de voz usando um ou mais padrões como G.711, G.722, G.723.1, G.728 e G.729.
- *codec* vídeo – do mesmo modo que o *codec* de áudio, faz a codificação de sinais de vídeo vindos de uma câmera e decodificação dos sinais vindos da

rede para sinais apropriados para representação em um monitor. Os algoritmos utilizados são o H.261 e H.263.

- H.225.0 RAS (*Registration, Admission and Status*) – o canal RAS transporta mensagens usadas pelos *gatekeepers* nos processos de registro de *endpoints* (terminais e *gateways*), que associam um número de telefone E.164 com o endereço/porta TCP/IP a ser usado para sinalização. O canal RAS também é usado para transmissão de admissão, mudanças de largura de banda, status e mensagens de desconexão entre *endpoints* e o *gatekeeper*. O protocolo H.225.0 RAS opera sobre UDP, o que requer que se inclua controle de *timeouts* e retransmissão dentro do protocolo de aplicação.
- H.255.0 Sinalização de chamada – este canal, conhecido em inglês como *Call Signaling Channel*, transporta mensagens de controle de chamada (sinalização) usando TCP, sendo portanto um canal confiável.
- H.245 Canal de Controle ou *Control Function* – este canal carrega mensagens de controle fim-a-fim que governam a operação das entidades H.323 (terminais, *gateways* ou *gatekeepers*). A função principal do canal H.245 é troca de funcionalidades ou capacidades. Outras funções incluem abertura e fechamento de canais lógicos, mensagens de controle de fluxo e comandos e indicadores de uso geral. Opera sobre TCP.
- RTP (*Real Time Protocol*) – protocolo de camada de aplicação, fornece serviços para entrega fim-a-fim para dados com requisitos de tempo real. Inclui *timestamping* (registro do tempo ou “hora”), seqüência de numeração, monitoração de entrega, multiplexação e controle de erros (*checksum*). Usado comumente sobre UDP, motivo pelo qual agrega funções típicas de transporte confiável.

- RTCP (*Real Time Control Protocol*) – acompanha o RTP, monitorando a qualidade da entrega dos dados através de relatórios de feedback.

A Figura 3 ilustra a pilha de protocolos H.323.

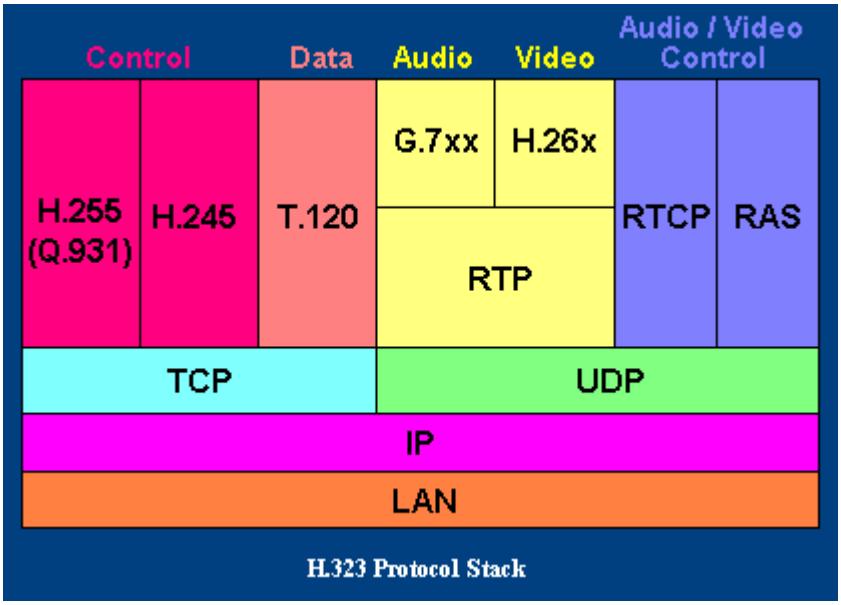


Figura 3: Pilha de protocolos H.323 (fonte http://www.pulsewan.com/data101/ h323_basics.htm).

Pela análise da figura, observa-se que VoIP opera em RTP sobre UDP, sobre IP. Evita-se o uso do TCP devido às características de controle de congestionamento e erro do TCP, que agem sobre a taxa de transmissão diretamente, sem intervenção ou controle por parte do protocolo de aplicação. Isso torna o TCP não adequado para comunicação em tempo real como transmissão de voz, pois essas características podem causar atrasos excessivos. O serviço UDP, sendo não orientado a conexão e não confiável, permite que o controle da confiabilidade e atraso seja transferido para a aplicação, que pode então decidir a melhor atuação de acordo com os requerimentos de qualidade. Em adição, o uso de UDP consente também a operação em *multicast/broadcast*. O protocolo RTP, por sua vez, não reserva recursos e não garante qualidade de serviço. O protocolo RTCP o faz através da monitoração do link, mas a maioria das aplicações VoIP funcionam transmitindo dados continuamente (*stream*) em pacotes, sem atenção a perdas ou atraso.

2.2.1. Atraso

Os atrasos em VoIP são causados principalmente devido a processamento, propagação e congestionamento. O processamento consiste na codificação/decodificação e encapsulamento/desencapsulamento. A propagação pela rede dos pacotes não ocorre instantaneamente, por isso há um atraso devido a ela, correspondente ao meio físico utilizado (fios metálicos, fibra ótica, ondas eletromagnéticas). E, por fim, o congestionamento na rede provoca enfileiramento de pacotes, acarretando maior tempo para chegada no destino.

A codificação e encapsulamento resultam em demoras maiores que as sentidas em sistemas de comutação de circuitos terrestres (sistemas que se utilizam de satélites apresentam um retardo significativo e bastante aparente). *Codecs* de voz apresentam taxas de codificação de saída na faixa aproximada de 5 a 64 kbps. De forma geral, quanto mais baixa a taxa de saída, mais complexo é o *codec* (pois há uma maior "economia" de dados gerados). Entretanto, a construção dos pacotes envolve um compromisso entre eficiência do *payload* (tamanho do *payload* dividido pelo tamanho total do pacote) e atraso do encapsulamento (tempo requerido para "encher" um pacote). Ou seja, quanto maior o *payload* de um pacote, mais eficiente ele é, pois há uma maior quantidade de dados úteis em relação a dados intrínsecos ao pacote que são irrelevantes à aplicação (o cabeçalho). Porém, maior o tempo gasto para completar este pacote e finalmente transmiti-lo.

Para IPv4, o cabeçalho completo RTP/UDP/IP mede 40 bytes. Um *payload* de 40 bytes significaria, portanto, uma eficiência de 50%. Com um *codec* de 64 Kbps, leva-se 5 ms para acumular 40 bytes. Com um *codec* mais avançado de 8 Kbps, por sua vez, leva-se 40 ms para acumular os mesmos 40 bytes. Um atraso de 40 ms é significante, portanto vários sistemas VoIP usam pacotes de 20 ms quando a voz está sendo codificada por *codecs* de baixa taxa, apesar da baixa eficiência de *payload* resultante.

Para conversação contínua, a capacidade de transmissão requerida para a chamada, *BW* (em Kbps), é relacionada ao tamanho do cabeçalho, *H* (em bits), a razão ou taxa de saída do *codec*, *R* (em Kbps), e o tamanho do *payload* em tempo de amostragem, *S* (em milissegundos)[4], pela seguinte equação:

$$BW = R + \frac{H}{S} \quad (1)$$

A fim de melhorar a eficiência do *payload*, há vários algoritmos de compressão do cabeçalho. O cabeçalho típico de 40 bytes do RTP/UDP/IP pode ser comprimido para um tamanho entre 2 a 7 bytes. Os menores requerimentos de *BW* demandam um grande atraso de encapsulamento e os *codecs* mais complexos. Desta maneira, é preciso haver um compromisso de engenharia de modo a atingir atraso de encapsulamento, complexidade do *codec* e largura de banda de chamada (*BW*) aceitáveis. A recomendação G.114 do ITU-T (*International Telecommunications Union – Telecommunications Standardization Sector*) relaciona os seguintes limites de tempo (Tabela 2) para uma transmissão em sentido único, com eco controlado [GOODE, 2002]:

Tabela 2: Recomendação G.144 ITU-T para VoIP [GOODE, 2002].

Atraso (sentido único)	Característica
0 a 150ms	Aceitável para a maioria das aplicações do usuário (atraso não detectável).
150 a 400ms	Aceitável para conexões internacionais (aqui começa-se a perceber o atraso como uma "hesitação" na resposta do interlocutor).
> 400ms	Inaceitável (a não ser em casos excepcionais). Aqui o atraso torna-se óbvio para os usuários e a comunicação resulta talvez impraticável, pois os usuários tendem a constranger-se e restringir a fala, de modo a evitar interrupções.

[4] Quanto maior o tempo de amostragem, menos amostragens por segundo, e vice-versa.

Acima de 250ms, a superposição de conversação (o problema de um interlocutor começar a falar enquanto o outro ainda está falando) torna-se significante.

A variação do atraso, conhecida como *jitter* (variação do tempo de interchegada dos pacotes), é também importante. O telefone ou *gateway* destino precisa compensar essas variações de atraso com um *jitter buffer*. Este *buffer* atua impondo um atraso nos primeiros pacotes e um atraso menor nos pacotes subseqüentes, de modo que a voz decodificada flua no receptor numa taxa constante. Pacotes que apresentem variação de atraso maior que a largura[5] do *jitter buffer* são descartados. Assim, se se deseja uma baixa perda de pacotes, a largura do *jitter buffer* é a máxima variação de atraso esperada. O atraso provocado pelo *jitter buffer* deve ser incluído no cômputo do atraso total fim-a-fim que um usuário experimenta durante uma conversação usando VoIP. A Tabela 3 ilustra os efeitos perceptivos do *jitter* em VoIP [MIRAS, 2002].

Tabela 3: Efeitos perceptivos na qualidade causados pelo *jitter*, para VoIP [MIRAS, 2002].

Atraso (sentido único)	Característica
< 40ms	*Jitter* não perceptível.
40 a 75ms	Boa qualidade, mas atrasos ou desordem ocasionais percebíveis.
> 75ms	Inaceitável; conversação torna-se excessivamente desordenada.

2.2.2. Modelagem da Fonte VoIP

Uma única fonte de voz é bem representada por um processo de dois estados [SCHWARTZ, 1996]. A conversação humana consiste de uma seqüência alternada de intervalos ativos, chamados *talk spurts* (“jorros” ou “esguichos”), seguidos de intervalos de

[5] Compreenda-se como “largura” do *jitter buffer* o maior atraso de tempo que este pode impor a um pacote.

silêncio. O silêncio não compreende somente os intervalos após as sentenças, mas silêncios entre palavras (como na língua escrita) e silêncios entre fonemas. Os intervalos de fala correspondem aproximadamente a 40% do tempo total, enquanto que os períodos de silêncio compreendem 60% do total de tempo da conversação, estatisticamente. (Intuitivamente, numa conversa entre dois interlocutores, pode-se estimar que cada um falará por cerca de metade do tempo total da conversação. Considerar que a comunicação dá-se através de duas linhas de mão única, uma que leva a conversação de A para B e outra que leva de B para A. Então, sob o ponto de vista de uma dessas linhas, haverá conversação durante 50% do tempo, e silêncio durante 50% do tempo total – período enquanto o interlocutor está calado, ouvindo a outra parte. Os silêncios entre fonemas e palavras contribuiriam pra reduzir o tempo total de fala para 40%.)

Os períodos de fala ativa e silêncio seguem uma distribuição estatística, com razoável aproximação, da distribuição de uma função exponencial. (A duração do *talk spurt* ou período ativo é bem aproximado pela distribuição exponencial; o intervalo de silêncio é menos bem representado por esta distribuição. Outros modelos com foram propostos com adição de estados a fim de melhorar a representação, mas isso a torna mais complexa.) Durante o período ativo, portanto, os *codecs* estão em ação e cria-se uma geração de dados constante, equivalente a uma fonte de fluxo CBR (*Constant Bit Rate*, ou taxa de bit constante), com taxa igual à do *codec* e com pacotes de tamanho fixo. Durante o intervalo de silêncio, não há geração de dados (desconsidera-se aqui dados de controle eventualmente passados entre os *codecs* transmissor e receptor). Assim, há um período de transmissão de dados, que corresponde ao período ativo, e um período inerte. Uma representação esquemática deste tipo de fonte está na Figura 4.

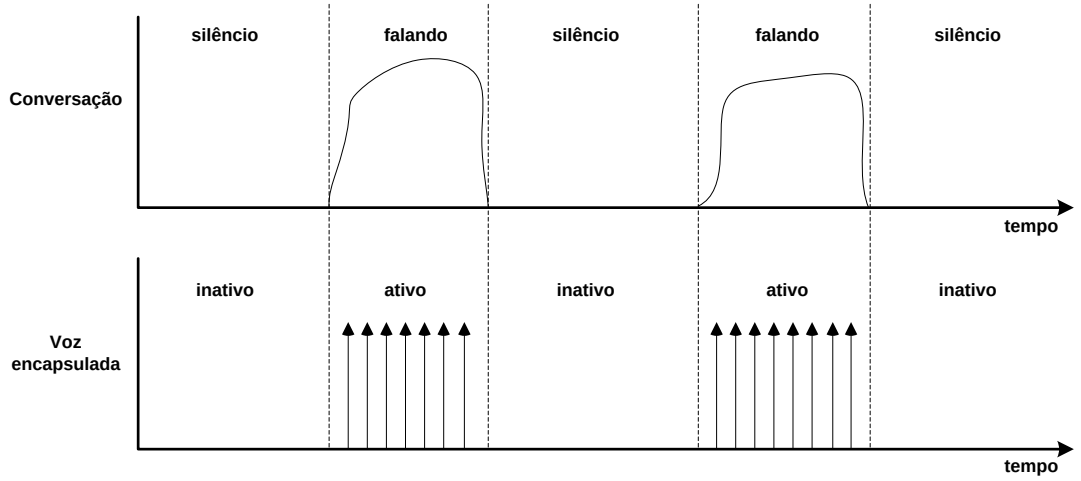


Figura 4: Comportamento de uma fonte de voz típica. Durante os períodos ativos, ou seja, quando o interlocutor está falando, o *codec* gera pacotes de tamanho fixo em intervalos de tempo regulares, indicados pelas setas verticais. Nos períodos de silêncio, não há geração de pacotes.

Esta distribuição, que apresenta um período ativo e um período inativo, ambos com variáveis aleatórias com médias de distribuição exponencial, recebe o nome especial de **Distribuição Exponencial On-Off** [TAQQU, 1997]. Ela é, portanto, completamente determinada através de duas variáveis aleatórias: a média do tempo On, e a média do tempo Off. Observar que, se o tempo On for levado bastante próximo de zero, de modo que, durante o estado On, somente haja a geração de um único evento ou "pacote" de voz, a distribuição exponencial On-Off reduz-se à distribuição exponencial correspondente ao modelo de Poisson (processo markoviano de nascimento e morte limitado a um elemento).

O modelo para a voz aqui descrito é então um modelo de nascimento e morte de dois estados (Figura 5). O parâmetro λ representa a taxa de transição para fora do estado de silêncio, e o parâmetro α é a taxa de transição para fora do *talk spurt* [SCHWARTZ, 1996]. Então, a média para o período ativo é $1/\alpha$ segundos, e o intervalo médio de silêncio é $1/\lambda$.

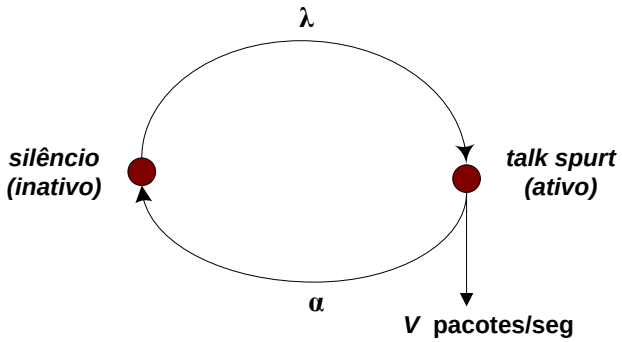


Figura 5: Modelo de dois estados para fonte de voz.

Para completa modelagem de uma fonte de voz exponencial on/off, tomam-se os seguintes parâmetros:

- *Burst*: média do tempo ativo ou *talk spurt*, enquanto a fonte está enviando pacotes. A literatura indica um valor entre 0,4 e 1,2 segundos (usualmente 0,4 segundos).
- *Idle*: média do tempo inativo ou silêncio, ou seja, fonte não está gerando pacotes. Entre 0,6 e 1,8 segundos, usualmente 0,6 segundos.
- *Packetsize*: tamanho dos pacotes, fixo, gerados e enviados pela fonte de voz durante os períodos ativos. Depende do *codec* utilizado.
- *Rate*: representa a taxa de envio de dados durante os períodos ativos. Para o padrão PCM, esta taxa é de 64 Kbps.

Durante os momentos On, pacotes com tamanho fixo (definido no parâmetro *Packetsize*) são gerados e enviados segundo uma distribuição exponencial de média de tempo definida no parâmetro *burst*. O envio dá-se a uma taxa constante, definida no parâmetro *rate*, e de forma alternada com momentos de silêncio ou inatividade, também segundo uma distribuição exponencial de média indicada pelo parâmetro *idle*. A Tabela 4 traz um resumo destes parâmetros do modelo de voz segundo uma distribuição exponencial On/Off.

Tabela 4: Parâmetros de modelo para geração de tráfego de voz (exponencial On/Off).

Parâmetro	Significado	Valor Usual
Burst	Média tempo ON	0,4 – 1,2s
Idle	Média tempo OFF	0,6 – 1,8s
Packetsize	Tamanho pacote IP	Depende do *codec* utilizado
Rate	Taxa de envio de dados	64Kbps (PCM)

Alguns autores discutem que, para tráfego agregado de várias fontes de voz, o melhor modelo não seria usar várias fontes On/Off exponenciais, mas sim outras modelagens mais sofisticadas como a de Fluxo (*Fluid Flow Modeling*) [SCHWARTZ, 1996] e Weibull [CHUAH, 2002]. Estas abordagens fogem ao escopo deste trabalho, que tenciona mostrar o modelo mais tradicional.

Um exemplo de código em C++ para geração de tráfego exponencial On-Off está disponível no ANEXO B.

3. FONTE DE VÍDEO

Vídeo é uma aplicação de tempo real, com requisitos bastante estreitos de qualidade de serviço quanto a atraso fim a fim e *jitter*. A qualidade de imagem e perda eventual de dados, entretanto, admitem maior tolerância. O sinal de vídeo, no formato não comprimido, representa uma fonde de tráfego com taxa de bits (bps) contínua. Os quadros de vídeo, transmitidos a uma taxa típica de 30 quadros por segundo, são amostrados e quantizados, pixel[6] por pixel, resultando num número constante de bits/quadro e, por conseguinte, bits por segundo [SCHWARTZ, 1996].

Considerar, como exemplo, o padrão norte-americano de 500 x 500 ou 250.000 pixels por quadro, aproximadamente. Usando 256 níveis de quantização para uma escala de cinza de 8 bits, isso resulta em uma taxa de 8 x 250.000 = 2 Mbits/quadro, ou ainda 2 Mbits x 30 quadros = 60 Mbps para o sinal de vídeo em tons de cinza. Para um sinal colorido tipo RGB (três cores básicas vermelho, verde e azul), sem sinal de luminância, isso representa 3 x 60 = 180 Mbps de capacidade requerida para um canal de transmissão.

Partindo-se para tv em alta definição (HDTV), com um número de pixels por quadro na ordem 1920 x 1080 = 2.073.600, 30 quadros por segundo, 8 bits para cada cor primária, tem-se um requerimento de 1920 x 1080 x 30 x 8 x 3= 1.492.992.000 ou aproximadamente 1,5 Gbps para transmissão sem compressão.

---

[6] Um pixel, ou *picture element*, é a unidade mínima ou básica de uma figura. Quando maior o número de pixels numa dada figura,maior a resolução desta. O processo de amostragem e quantização designa um certo número de bits para cada pixel, de modo que as características de luminância e crominância destes pixels seja representada digitalmente.

3.1. VÍDEO NÃO COMPRIMIDO

A transmissão de vídeo por rede de computadores, sem compressão, requer uma largura de banda ainda muito alta para os padrões atuais, tomando por base o Ethernet de 10Mbps e o Fastethernet de 100Mbps, as duas tecnologias de rede mais em uso. Mecanismos de compressão tornam-se indispensáveis para transmissão de vídeo em redes de dados.

Da seção anterior, infere-se que uma fonte de vídeo não comprimido reduz-se a uma fonte CBR. Com efeito: se a fonte produz quadros de tamanho fixo, com quantidade fixa de pixels por quadro; e se cada pixel é descrito por uma quantidade fixa de bits, isto resulta numa quantidade fixa de bits por quadro. Se a fonte gera uma quantidade fixa de quadros por segundo, então multiplicando a quantidade de bits por quadro pela quantidade de quadros por segundo, ter-se-á o número de bits por segundo (bps) que tal fonte de vídeo não comprimido gera.

3.1.1. Modelagem da Fonte de Vídeo Não Comprimido

A modelagem para a fonte de vídeo não comprimido torna-se simplificada, frente às suas características. A fonte de tráfego CBR, com vazão em bps equivalente à fonte de vídeo não comprimido, basta para uma boa representação.

A parametrização desta fonte também é direta. Pode-se resumir em:

- Pixels por Quadro;
- Bits por Pixel;
- Quadros por Segundo.

A multiplicação dos dois primeiros parâmetros resulta na quantidade de bits por quadro da fonte. A multiplicação desta última pelo número de quadros por segundo resulta na

taxa de bits por segundo da fonte, que é constante. Pode-se também considerar somente esta última grandeza como único parâmetro da fonte de vídeo não comprimido.

3.2. MPEG

O padrão mais usado de codificação e compressão para vídeo é o proposto pelo *Moving Picture Experts Group*, e que recebe a mesma sigla – MPEG. O padrão MPEG-1 lida com compressão de sinais de vídeo para armazenamento em mídia digital, como CD-ROM's. Tem como meta gerar uma taxa de dados codificada de 1,5Mbps ou menos (incluindo áudio e vídeo). O MPEG-2 é uma codificação de alta qualidade, incluindo a usada para HDTV.

O padrão MPEG é dividido em 5 camadas, a saber, em ordem de tamanho:

1. GoP (*Group of Pictures*), ou Grupo de Figuras ou Quadros.
2. Quadro ou *Picture*.
3. *Slice* ou fatia.
4. Macrobloco.
5. Bloco.

A entrada do codificador MPEG consiste em uma série de quadros de vídeo, cada quadro sendo uma imagem estática contendo uma matriz bidimensional de elementos de quadro (pixels). Na codificação, para cada pixel, será armazenada informação de luminância e crominância. O algoritmo de compressão procura reduzir a taxa de dados focando a redundância espacial e temporal, existentes em vídeos. A redundância espacial aproveita-se da similaridade entre pixels que ocorrem numa mesma vizinhança (Figura 6) [SANTOS, 2003].

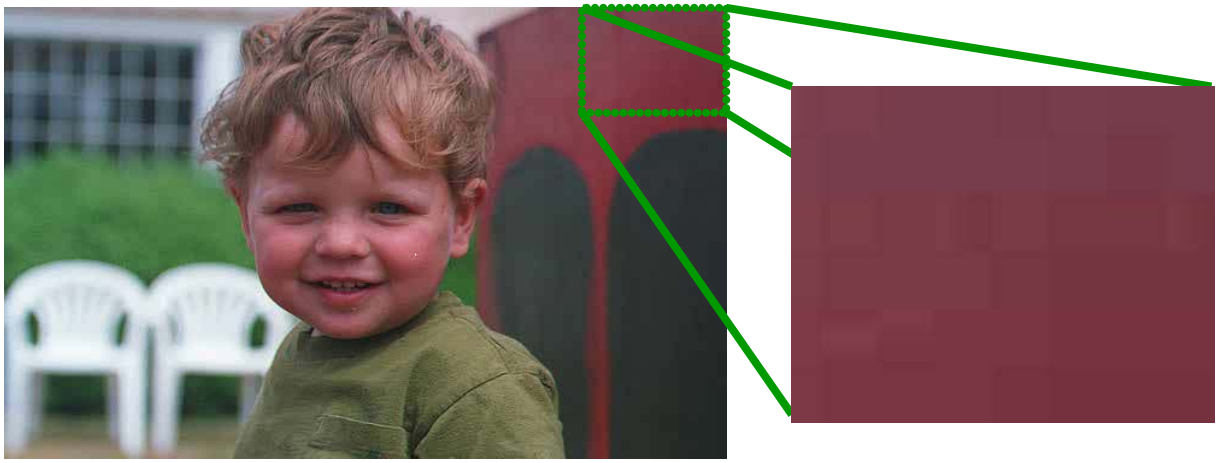


Figura 6: Redundância espacial. O destaque mostra um grupo de pixels similares na imagem [SANTOS, 2003].

As redundâncias espaciais são reduzidas através de transformadas (como a DCT – *Discrete Cosin Transform*, ou transformada discreta de cosseno) e codificação por entropia (*entropy coding*).

A redundância temporal é a similaridade entre quadros sucessivos de um vídeo. Imagine-se, por exemplo, uma cena parada, de telejornal, onde somente o apresentador fala. De um quadro para outro, praticamente só os movimentos faciais do apresentador mudarão, permanecendo constantes os demais elementos da cena, como o fundo (Figura 7).

Figura 7: Redundância temporal. As mudanças entre os quadros 950, 951 e 952 resumem-se basicamente a movimentos faciais, enquanto que o resto da cena permanece inalterado [SANTOS, 2003].

As redundâncias temporais são reduzidas através de predição de quadros futuros, baseada em vetores de movimento. Isso é conseguido por intermédio da geração, pelo codificador MPEG, de três tipos de quadro:

- **I** (*Intraframe coded* ou intraquadro): são codificados com pequena compressão (somente compressão espacial), e não guardam referência de ou para nenhum outro quadro. Os quadros I são criados de maneira similar às imagens codificadas por JPEG, usando técnicas de DCT e quantização. São quadros âncora, e permitem acesso randômico às seqüências de vídeo [CHODOREK, 2000]. Por não conter referências de ou a outros quadros, os quadros I são os maiores em termos de quantidade de dados, mas a degradação de qualidade de imagem é a menor. A seqüência de vídeo deve conter ao menos um quadro I. Através da inserção de vários quadros I na seqüência, permite-se a recuperação de perdas de dados (se uma determinada informação de quadro é perdida, todas as informações de quadro seguintes, baseadas nesta perdida, ficarão incorretas; ver demais tipo de quadro adiante).
- **P** (*Predictive coded* ou preditivos): codificados usando predição com compensação de movimento, usando referências do quadro anterior de tipo I ou P. Os quadros P podem ser usados em predições subseqüentes. Para que um quadro P seja decodificado, o quadro precedente, de tipo I ou P, tem de haver sido decodificado corretamente (caso contrário, as referências serão falhas). O tamanho do quadro P é o segundo maior, e a qualidade de imagem, a segunda melhor entre os três tipos de quadro. A compressão, pois, é tipicamente maior que aquela obtida para quadros I [CHODOREK, 2000; WAKAMIYA, 2000].
- **B** (*Bi-directional predictive coded* ou bidirecional): codificados usando predição com compensação de movimento, usando referências preditivas tanto

> do quadro anterior como do seguinte, de tipo I ou P. Quadros B não são usados em predições subseqüentes, e têm a maior compressão de todos os três tipos. A qualidade de imagem, por conseguinte, também é a menor. O reordenamento de quadros faz-se necessário quando se introduz quadros tipo B, pois um quadro futuro que lhe serve de referência tem de ser codificado e decodificado antes do quadro B, causando atrasos extras.

Depois da codificação, os quadros resultantes são reunidos numa seqüência periódica, determinística, que é denominada GoP (*Group of Pictures*), ou Grupo de Quadros. A especificação MPEG não define quais tipos de quadros (I, P ou B) devem compor a estrutura GoP, ou o comprimento desta seqüência. Apenas especifica que o quadro inicial deve ser do tipo I, e a seqüência pode até mudar no tempo. Para MPEG-2, a estrutura GoP é inclusive opcional, mas ainda largamente usada de modo a permitir acesso aleatório ao vídeo e conferir maior tolerância a erros.

Codificadores típicos usam estruturas com um padrão de GoP fixo. O tamanho do GoP, ou seja, o número de quadros num GoP, é especificado pelo parâmetro *N*, e a distância entre quadros I e P é especificada pelo parâmetro *M*. Ver ilustração de uma estrutura GoP na Figura 8.

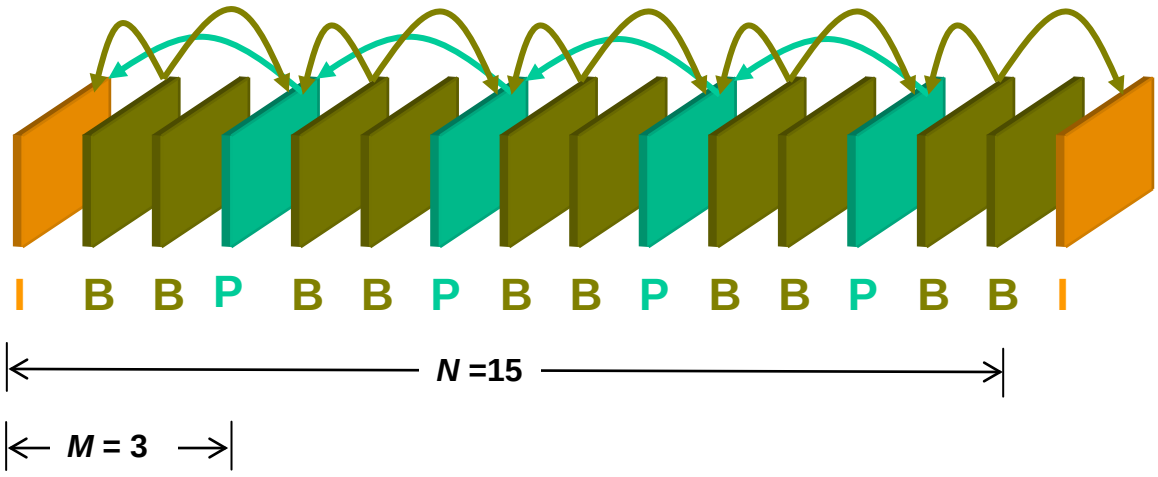


Figura 8: Estrutura GoP de tamanho 15 (*N* = 15) e distância entre I e P, *M*, igual a 3. As setas mostram as referências preditivas de cada quadro [SANTOS, 2003].

Cada quadro consiste em um número de fatias ou *slices*, e cada *slice* é composto de macroblocos, de tamanho 16 x 16 pixels cada. A transformada de cosseno discreta, DCT, é então aplicada aos macroblocos e cada coeficiente da DCT é então quantizado de acordo com uma escala específica. A Tabela 5 traz os parâmetros da codificação MPEG (para formato PAL – *Phase Alternating Line* ou SECAM – *Système Electronique Couleur Avec Mémoire*)[7] e a Tabela 6, números típicos obtidos pela compressão.

Tabela 5: Parâmetros típicos MPEG [SANTOS, 2003].

Parâmetro	Valor
Resolução	384 x 288
Fator de Quantização	8
Quadros entre figuras I	5
Quadros entre figuras P	2
GoP	...IBBPBBI...

[7] Para formato NTSC (*National Television System Committee*), usado nos EUA e alguns outros países, costuma-se adotar a resolução de 352 x 288.

Tabela 6: Desempenho típico da compressão MPEG [SANTOS, 2003].

Tipo de Quadro	Tamanho (Kbytes)	Taxa de Compressão
I	18	7:1
P	6	20:1
B	2,5	50:1
Média	**4,8**	**27:1**

Escolhas de *N* e *M* afetam a qualidade de vídeo percebida e as características de tráfego gerado. Imagine-se uma seqüência de vídeo sem qualquer compressão. Conforme já visto no início deste capítulo, os quadros de vídeo não comprimidos têm exatamente o mesmo tamanho, gerando, pois, um tráfego constante em termos de bits por segundo.

Para efeito comparativo,a traz as larguras de banda requeridas pelos formatos de vídeo mais comumente usados [MIRAS, 2002].

Tabela 7: Largura de banda utilizada pelos formatos de vídeo mais usados [MIRAS, 2002].

Formato de Vídeo	Requerimento Típico de Largura de Banda
HDTV não-comprimido	1,5 Gbps
HDTV ínterim	360 Mbps
Standard Definition TV (SDTV), SMPTE	270 Mbps
MPEG-2 4:2:2 comprimido	25 – 60 Mbps
HDTV em MPEG-2	19,4 Mbps
SDTV em MPEG-2	6 Mbps
MPEG-1	1,5 Mbps
MPEG-4	5 Kbps – 4 Mbps
H.323 (H.263)	28 Kbps – 1 Mbps

3.2.1. Modelagem da Fonte MPEG

Para um vídeo comprimido no padrão MPEG percebe-se que, se os tamanhos de quadros são distintos, então a taxa de tráfego deve flutuar, quadro a quadro, dependendo do tipo de quadro exibido naquele momento. Durante a transmissão dos quadros B e P, a taxa é mais baixa, atingindo picos de tráfego na transmissão dos quadros I, os maiores. Esta é a

característica chave do tráfego gerado por uma fonte de vídeo comprimido MPEG: **taxa variável de bit,** ou **VBR (*Variable Bit Rate*).**

Este tipo de comportamento é na verdade fruto da codificação MPEG do tipo VBR. Nesta, a escala de quantização usada para os coeficientes DCT é idêntica para todos os macroblocos na seqüência de vídeo. Como resultado, a seqüência codificada tem uma qualidade de vídeo relativamente constante, mas gera um tráfego de rajada (*bursty*) bastante forte. Na codificação MPEG do tipo CBR, aplica-se controle de taxa, que ajusta a escala de quantização de acordo com a largura de banda alocada e o nível de ocupação dos *buffers* [WAKAMIYA, 2000]. Neste caso, o tamanho desejado para o GoP é primeiro determinado a partir da largura de banda disponível. Então, o tamanho de cada quadro no GoP é definido. Na quantização para cada quadro, escala de quantização muda dinamicamente (no nível do macrobloco), de modo a que o quadro resultante tenha o tamanho de bits definido. A média da taxa de tráfego final, para o GoP, aproxima-se então da largura de banda disponível. Ainda assim, devido à natureza dos diversos tipos de quadro, a flutuação da taxa entre quadros é inevitável. Em adição, a qualidade de imagem dos quadros varia entre quadros e dentro de um mesmo quadro, devido ao controle da quantização.

A característica variável do tráfego VBR pode ser vista na Figura 9. Trata-se do gráfico Largura de Banda versus Tempo (em quadros) medida por [GARRETT, 1994] para o filme *Star Wars* (Guerra nas Estrelas)[8], de duas horas, codificado em preto-e-branco por um método simplificado de MPEG intraquadro. No gráfico, podem-se observar três picos de tráfego próximos ao centro. Estes correspondem a cenas com efeitos especiais visuais,

[8] Este filme é largamente e habitualmente utilizado pelos autores nos estudos de codificação de vídeo e características de tráfego de vídeo em rede, e arquivos com a seqüência codificada (*tracing*), originalmente produzida pela Bellcore, estão disponíveis em vários sites na Internet, como por exemplo http://www.research.att.com/~breslau/vint/trace.html e ftp://thumper.bellcore.com/pub/vbr.video.trace.

contendo componentes de alta freqüência espacial (largas mudanças na cena em pouco tempo). Especificamente, as cenas são “salto para o hiperespaço”, a explosão do planeta Alderaan, destruído pela Estrela da Morte (*Death Star*), e o “salto do hiperespaço”.

Figura 9: Variação típica da taxa de tráfego de uma fonte VBR, aqui mostrada para a seqüência codificada, em preto-e-branco, de duas horas do filme *Star Wars* [GARRETT, 1994].

3.2.2. Geração Do Tráfego MPEG – VBR

Para fins de modelagem estocástica do tráfego de vídeo VBR, a literatura classifica-o como sendo do tipo *auto-similar* [TAQQU, 1997]. A auto-similaridade (*self-similarity*) é um fenômeno conhecido e bastante observado na natureza. Significa que a estrutura básica de um objeto ou observação pode ser encontrada, invariável, em diversas escalas de tempo (não por coincidência, esta também é uma das definições do *fractal*[9]).

A modelagem do tráfego de fonte VBR não é trivial, devido à complexidade do processo estocástico da seqüência de vídeo. Há várias propostas de modelagem, dentre:

- Processo de Poisson Modulado por Markov (MMPP) [SCHWARTZ, 1996];

[9] Fractais serão abordados adiante neste trabalho.

- Processo Auto-Similar Modulado por Markov [LIU, 1999a; LIU, 1999b];
- Método do Fluxo Contínuo Estocástico (*Stochastic Fluid Flow – SFF*) [SCHWARTZ, 1996];
- Pseudo auto-similar [KHAYARI, 2002];
- Processo de Pareto On/Off [GORDON, 1995];
- Processo Vetorial Auto-regressivo (*VAR – Vector Auto-Regressive process*) [SHIN, 1995]
- Transformada de Wavelet ou modelagem no Domínio de Wavelet [MA, 1998a; MA, 1998b; WANG, 1997]
- Modelo multifractal multiplicativo [KRISHNA, 2002];
- Movimento Browniano Fracional (fBM – *Fractional Brownian Motion*) (MANDELBROT, 1969);
- Método TES – *Transform-Expand-Sample* [MATRAWY, 2002; MELAMED, 1994];
- Modelo de Rayleigh Autoregressivo Composto (RAR – *Composite Rayleigh Auto-Regressive*) [XUYONG, 2000].

A geração do tráfego VBR, por simulador, pode ser realizada espelhando-se um tráfego real, coletado por um analisador ou monitor. Em outras palavras, usando-se um *trace* de um tráfego de uma fonte VBR real. *Trace* é um histórico, um registro ou memória, contendo medições e/ou descrições feitas de um determinado evento ou sistema, no tempo, em seqüência cronológica. Um *trace* de um tráfego real conteria, por exemplo, o volume de dados ou ocupação da banda disponível medida em intervalos de tempo fixos. Para gerar este mesmo tráfego novamente, um gerador leria o arquivo de *trace* e emitiria o mesmo número de dados medidos e gravados ali, seguindo a seqüência temporal. A desvantagem deste método é que a geração é fixa e não admite parametrização.

A simulação também pode empregar métodos numéricos, baseados nas modelagens estocásticas relacionadas anteriormente. Uma abordagem simples é usar uma combinação de vários geradores do tipo Pareto On/Off, que, segundo [TAQQU, 1997; GORDON, 1995], resulta em tráfego auto-similar, VBR. O ANEXO C traz um exemplo de código usando esta abordagem e também um segundo, usando a técnica TES para simulação de tráfego MPEG-4.

4. FONTES DE TRÁFEGO AUTO-SIMILAR OU FRACTAL

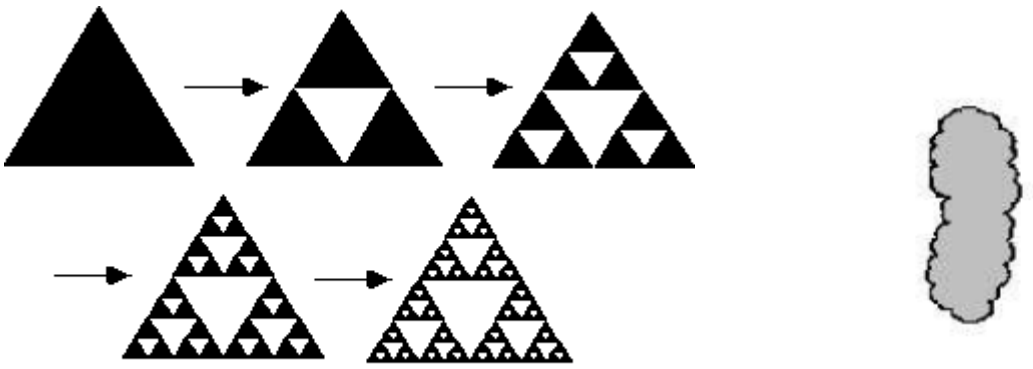
O termo *auto-similar* (ou *self-similar*, em inglês) significa que a caracterização estatística do fenômeno é essencialmente invariante com a escala do tempo. As mesmas propriedades estatísticas podem ser vislumbradas se a escala de tempo muda de horas para minutos, para segundos, para centésimos de segundo. Este fenômeno de auto-similaridade foi primeiro observado pelo matemático polaco-francês Benoit Mandelbrot (MANDELBROT, 1983 *apud* SCHWARTZ, 1996), que observou este tipo de comportamento surgindo não somente nas séries de tempo geradas por tráfego, como também em vários fenômenos naturais. Coube também a Mandelbrot cunhar a palavra “fractal”.

4.1. FRACTAIS

Matemáticos como Cantor, Koch, Hausdorff e Sierpinski já cogitavam a idéia de objetos ou figuras de dimensão fracionária, desde o século 19. A denominação “fractal”, montada por Mandelbrot, deriva do latim *fractus*, significando “fração”, “fragmento”, “fragmentado” [CRUZ, 2003]. Dentre as várias definições de fractal, a mais simples, e a de interesse neste trabalho, é a que remete à condição de auto-similaridade ou auto-afinidade geométrica do objeto. Um fractal, desta maneira, é uma forma geométrica que se mostra invariante quando observada em escalas diferentes.

Uma das figuras mais usadas para visualização da auto-similaridade fractal é o Triângulo de Sierpinski (Figura 10a). É constituído por mini-triângulos que são, por sua vez, cópias perfeitas da figura completa. Sua construção parte de um triângulo qualquer. Usa-se então os pontos médios de cada lado como vértices de um novo triângulo, e remove-se este triângulo do original. O processo continua, resultando na remoção de triângulos cada vez menores, até que microscópicos.

Observando a figura de qualquer escala, vê-se um objeto idêntico à figura como um todo. Este é um caso de um fractal **auto-similar exato**, ou simplesmente auto-similar [CRUZ, 2003]. Há fractais que, apesar de serem formados também por mini-cópias do objeto completo, não mantém fixas as proporções originais para estas cópias (são anisotrópicas). Ao mudar-se a escala de observação, o tamanho das cópias não se altera uniformemente em todas as direções. Estes fractais são chamados **auto-afins.** A Figura 10b ilustra o caso: o contorno da nuvem é formado por cópias não uniformes do todo. Estatisticamente, a fronteira da nuvem mantém uma correlação estatística quando observada em diferentes escalas.



(a) **(b)**

Figura 10: (a) Construção do Triângulo de Sierpinski (Fonte: http://math.bu.edu/DYSYS/chaos-game/node2.html). (b) Contorno de nuvem revelando um fractal auto-afim [CRUZ, 2003].

Os fractais estão presentes na natureza em exuberância. Nuvens, rochas, perfil da costa de um continente e, como justificativa para sua abordagem neste trabalho, no comportamento estatístico de alguns tipos de fontes de tráfego em redes de computadores multi-serviço, que apresentam auto-similaridade. A denominação auto-similar, *self-similar* ou fractal é aplicado a estas fontes.

4.2. AUTO-SIMILARIDADE EM TRÁFEGOS DE REDES MULTI-SERVIÇO

[HUANG, 1995a] relata que medições extensivas feitas sobre tráfego real em redes de serviços integrados (ISDN) levaram à conclusão que o tráfego Ethernet não pode ser suficientemente representado por modelos tradicionais (baseados em Poisson ou processos Markovianos). O modelo auto-similar ou fractal seria mais preciso nestas representações. Em adição, o tráfego VBR, o tráfego WAN, o tráfego gerado pelas transferências da *World Wide Web* (WWW) e o tráfego de rede local (LAN - *Local Area Network*) também exibiriam características de auto-similaridade [LEDESMA, 2000; CROVELLA, 1997]. Da mesma forma, estudos realizados em tráfego agregado de rede local por [LELAND, 1994] nos laboratórios da Bellcore Morristown Research and Engineering Center[10], entre agosto de 1989 e fevereiro de 1992, mostram o comportamento condizente com auto-similaridade (ou invariância com a escala de tempo) e distante dos modelos convencionais baseados em telefonia e Poisson.

Em suma, os estudos produzidos revelaram [PRASAD, 1996]:

1. o tráfego real de redes é claramente distinto do tráfego sintetizado pelos modelos teóricos tradicionais;
2. o tráfego de redes é estatisticamente auto-similar, i.e., as rajadas distribuem-se por diversas escalas de tempo, desde milissegundos até horas;
3. o tráfego é mais explosivo do que o esperado; a agregação de várias fontes ou fluxos de tráfego intensifica a explosividade (*burstiness*), o que contrasta com

[10] A topologia consistia em cerca de 140 *hosts* e roteadores conectados na rede local. As medições foram feitas por um período de 27 horas, durante o qual 121 equipamentos estavam em operação. Maiores detalhes estão disponíveis em [LELAND, 1994].

> o conceito de que a multiplexação de um grande número de fontes independentes de tráfego resultaria em um tráfego uniforme (Poisson).

Colocando as medições de tráfego num gráfico, onde o eixo vertical indica o número de pacotes por unidade de tempo (a *vazão* ou velocidade instantânea) e o eixo horizontal, o instante de tempo, pode-se visualizar o aspecto de um tráfego auto-similar (Figura 11).

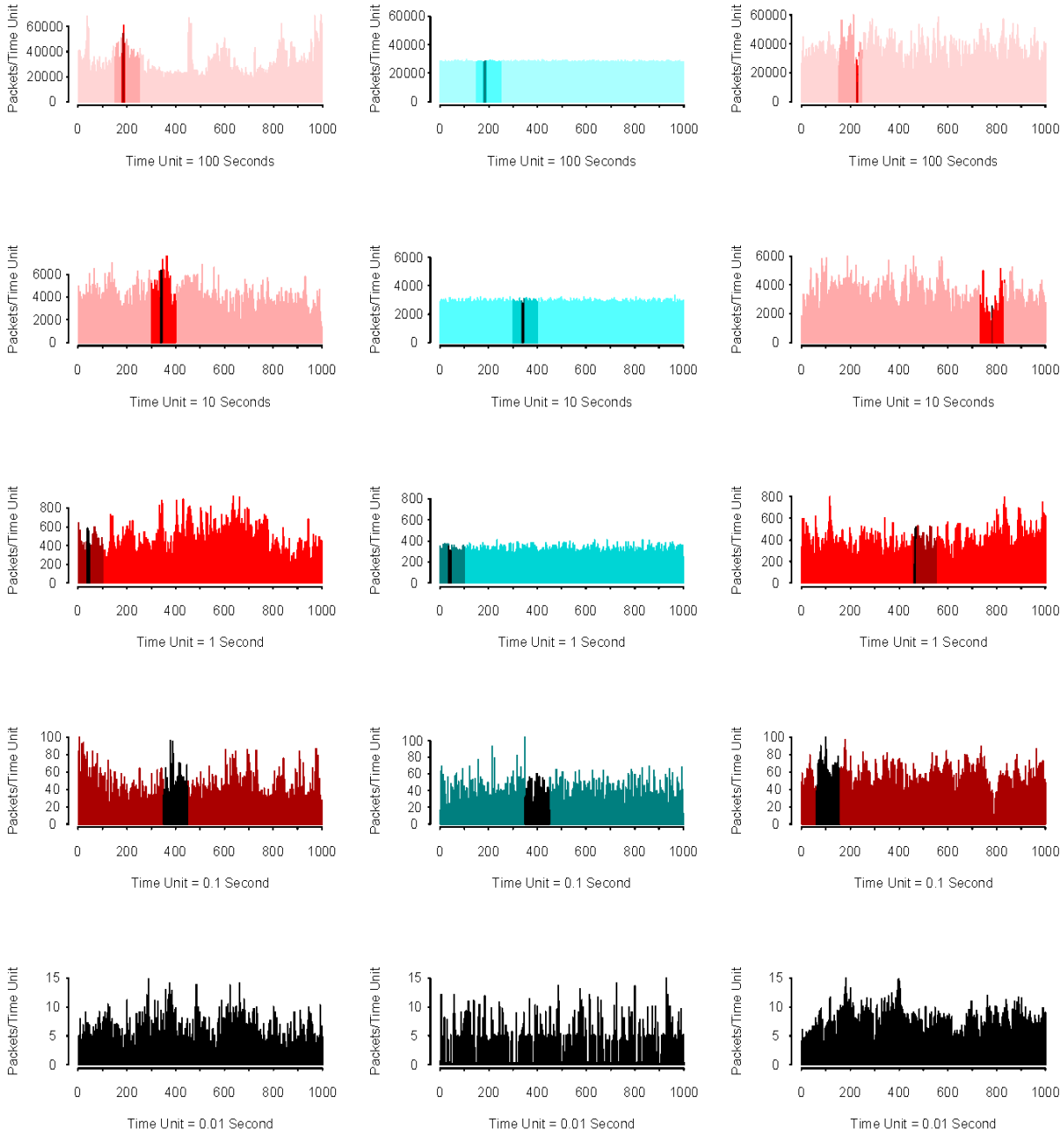


Figura 11: (Coluna esquerda) Tráfego real Ethernet; (coluna central) tráfego simulado gerado por modelagem tradicional (Poisson); (coluna direita) tráfego simulado gerado por modelagem auto-similar. Cinco diferentes escalas de tempo são usadas. As gradações de cor indicam os mesmos segmentos de tráfego nas diferentes escalas de tempo. Legenda: (eixo vertical) *Packets/Time Unit*: pacotes/unidade de tempo; (eixo horizontal) *Time Unit*: unidade de tempo. [TAQQU, 1997].

Na Figura 11, obtida de [TAQQU, 1997] e que segue a mesma metodologia de construção indicada em [LELAND, 1994], produziu-se uma seqüência de gráficos de contagem de pacotes (pacotes por unidade de tempo) para cinco unidades de tempo distintas. O primeiro gráfico da seqüência origina-se na escala de 100 segundos e cada gráfico

subseqüente é obtido do anterior, aumentando a resolução da escala de tempo por um fator de 10, até 0,01 segundo. A contagem de pacotes é obtida também do gráfico anterior, a partir de um subintervalo escolhido aleatoriamente, e indicado através de gradação de cor. A coluna da esquerda traz uma medição de tráfego real; a coluna central é fruto de tráfego simulado, gerado por uma modelagem tradicional baseada em Poisson; o tráfego da direita é também advindo de geração simulada, porém com modelo auto-similar [TAQQU, 1997].

4.3. MODELAGEM DA FONTE AUTO-SIMILAR

Com exceção talvez da primeira linha (que teria sofrido a influência do ciclo diurno de uso da rede de onde foram coletadas as medições), todos os gráficos da coluna esquerda (Figura 11) são similares entre si. Ou seja, o tráfego Ethernet aparenta ter o mesmo comportamento tanto em grande escala (minutos e horas), quanto em pequena escala (segundos, milissegundos). Esta característica de auto-similaridade é diferente do tráfego convencional de telefonia (que é usado como modelo também para redes) e de modelos estocásticos para redes baseados tipicamente em Poisson.

A coluna do meio ilustra falhas na geração de tráfego simulado baseado em modelos tradicionais de Poisson. Notadamente, neste modelo, o tráfego agregado em escala de tempo grande exibe um aspecto uniforme, menos explosivo, o que não condiz com a realidade demonstrada na coluna esquerda. As predições estatísticas para o modelo de Poisson levam também ao entendimento que o tráfego tende a se tornar tanto menos explosivo, ou uniforme, quanto maior for o número de fontes geradoras de tráfego. [LELAND, 1994] questiona também este conceito; a explosividade ou *burstiness* do tráfego em LAN tipicamente é intensificada frente ao aumento do número de fontes ativas de tráfego.

Finalmente, a coluna da direita, que representa os gráficos de tráfego sintetizado a partir de modelagem auto-similar, apresenta-se visualmente indistinta do comportamento real do tráfego medido.

Os processos auto-similares exibem a chamada *long range dependence* (LRD), ou dependência de longo alcance. Sua função de autocorrelação decai lentamente, não exponencialmente. Os modelos estocásticos tradicionais para fontes de tráfego em redes exibem, em contraste, *short range dependence* (SRD), ou dependência de curto alcance, o que significa que a função de autocorrelação destes decai exponencialmente [HUANG, 1995a]. Informalmente, a LRD significa correlações entre escalas de tempo grandes. A presença ou ausência de LRD é de importância no comportamento previsto por modelos analíticos para redes; a presença de LRD, por exemplo, muda totalmente a característica de tempo de espera em fila [PAXSON, 1997].

Interessante notar que o aspecto do tráfego de fonte VBR (Figura 9) também remete a uma aparência de auto-similaridade. Esta fonte, entretanto, além de exibir LRD, também possui estrutura fortemente SRD, o que sugere um modelo auto-similar assimptótico, no lugar de uma modelagem auto-similar exata [HUANG, 1995a; HUANG, 1997].

O desenvolvimento matemático para caracterização da auto-similaridade foge ao intuito deste trabalho. O leitor é convidado a consultar as referências, sugerindo-se [LELAND, 1994; PRASAD, 1996; HUANG, 1997; BERAN, 1995; DAUT, 1999].

4.4. GERAÇÃO DO TRÁFEGO AUTO-SIMILAR

Há vários métodos ou modelos para a síntese de processos auto-similares. Dentre os quais [LEDESMA, 2000]:

- *Fast Fractional Gaussian Noise* (fFGN), ou ruído gaussiano fracional, rápido, proposto por Mandelbrot;
- *Alternating Renewal Process*, ou processo de renovação alternada;
- Processos autoregressivos;
- Mapas caóticos;
- *Fractional Autoregressive Integrated Moving Average Process* (F-ARIMA), ou processo de média móvel integrada autoregressiva fracional;
- *Fast Fourier Transform* (FFT), ou Transformada Rápida de Fourier;
- *Fractal Renewal Process*, ou processo de renovação fractal;
- *Queueing Processes*, ou processos de filas;
- *Random Midpoint Displacement* (RMD), ou deslocamento aleatório de ponto médio;
- *Spatial Renewal Process*, ou processo de renovação espacial;
- *Quasi-self-similar Markov Modulated Rate Process*, ou processo de razão modulado por Markov, quase auto-similar;
- Transformada de Wavelet;
- *Successive Random Addition* (SRA), ou adição sucessiva aleatória [LINAWATI, 2002]

Os detalhes dos algoritmos de geração requerem profundidade matemática para sua compreensão, o que não é abrangido pelo escopo deste trabalho. O ANEXO D traz um exemplo de código em C para um gerador auto-similar. O programa gera ruído gaussiano fracional (FGN) baseado em algoritmo FFT (*Fast Fourier Transform*), resultando em comportamento auto-similar.

5. FONTE DO TIPO FILE TRANSFER PROTOCOL (FTP)

FTP, ou *File Transfer Protocol*, é um protocolo para transferência de arquivos de um *host* para outro. Numa sessão FTP típica, um usuário deseja transferir arquivos de seu *host* (chamado *host* local) para um *host* remoto, ou ainda arquivos do *host* remoto para o *host* local. O usuário interage com o protocolo FTP através de um agente, que por sua vez tem uma interface com o usuário [KUROSE, 2001]. A Figura 12 ilustra a operação.

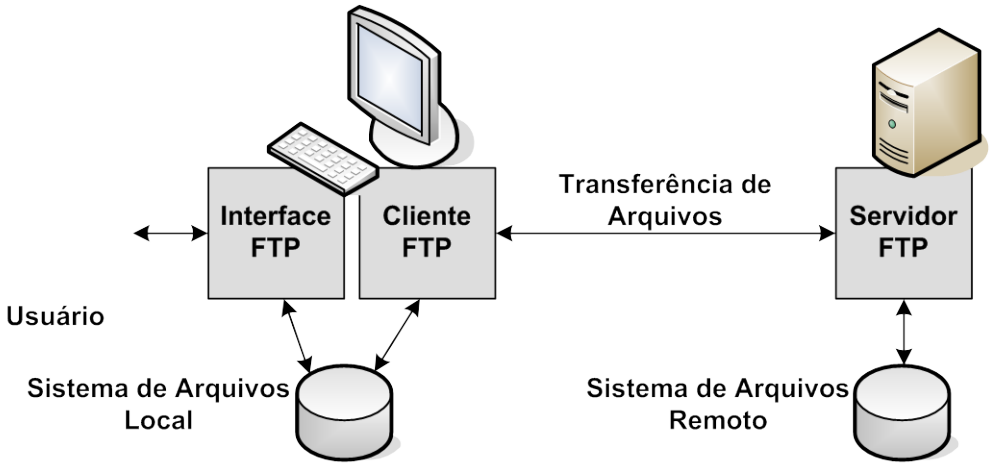


Figura 12: Transferência de arquivos entre sistemas de arquivo remoto e local.

FTP é um protocolo de aplicação e usa os serviços do protocolo de transporte TCP. Quando um usuário requisita a transferência de um arquivo de ou para um *host*, o FTP abre uma conexão de dados TCP na porta número 20, do servidor. O FTP então envia o arquivo através desta conexão, que é fechada após a transferência. Este procedimento simples e direto resultará no comportamento do tráfego gerado pela aplicação FTP.

5.1. MODELAGEM DA FONTE FTP

Quando o usuário aciona o comando de transferência de arquivo, o FTP abre uma conexão TCP entre os *hosts* envolvidos e executa a transferência. O procotolo TCP, por sua vez, é um protocolo de transporte orientado a conexão, fim a fim, que garante confiabilidade. Mais ainda, o TCP provê controle de fluxo (*flow control*), que evita que um *host* transmissor sobrecarregue os *buffers* de recepção de um *host* mais lento, e controle e prevenção de congestionamento (*congestion control* e *congestion avoidance*), que por sua vez regula a velocidade de transferência de segmentos de acordo com a situação de congestionamento instantânea da rede. Ou seja, o protocolo TCP mantém o controle sobre a vazão (*throughput*) da conexão formada, que será função – em essência – do congestionamento da rede, da incidência de erros, da velocidade de processamento individual dos *hosts* e da capacidade dos *links* físicos entre os *hosts*. O protocolo FTP enxerga tão somente uma conexão confiável do tipo *stream*, através da qual ele enviará ou receberá um fluxo contínuo de dados. Caberá ao TCP, portanto, estabelecer quão rápido este fluxo de dados poderá ser estabelecido.

O protocolo TCP, tipicamente, procurará transmitir os segmentos tão velozmente quanto permitir a rede (e o processamento dos *hosts*). Se não houver estouros de tempo (*timeouts*), não houver detecção de erros ou congestionamento, o protocolo TCP tenderá a atingir o limite de capacidade de transmissão do *link*, provendo a transferência mais rápida possível. Havendo tráfego concorrente na rede, o protocolo TCP detectará sua influência através de seus mecanismos de controle de erros e congestionamento e “sufocará” sua própria velocidade de transmissão. Se o tráfego concorrente também for de natureza TCP, os recursos da rede (ou seja, a banda) tenderão a se dividir de forma igualitária [KUROSE, 2001], pois todos efetuarão o estrangulamento da transmissão.

Mas, sendo o tráfego concorrente do tipo UDP, por exemplo, e sem controle de congestionamento por parte da aplicação, este poderia dominar a banda, pois o TCP cederá espaço para ele. Com efeito, o TCP tenderá a "acompanhar" o gráfico de banda consumida dos tráfegos concorrentes, devido à reação mais tardia do mecanismo de controle de congestionamento. Os picos de utilização dos tráfegos concorrentes serão seguidos por uma diminuição da taxa de transferência do TCP, e os momentos de baixa transferência dos tráfegos concorrentes estimularão maior velocidade no TCP. Assim sendo, é factível deduzir que o comportamento estatístico da transmissão TCP, considerando um fluxo de dados constante vindos da aplicação e um processamento também constante dos *hosts*, é em essência determinado pelas condições da rede; mais evidentemente, pelo comportamento dos tráfegos concorrentes.

5.2. GERAÇÃO DO TRÁFEGO FTP

Pelo que foi exposto anteriormente, infere-se que a aplicação FTP herdará do TCP seu comportamento de geração de tráfego. Isso não leva em conta o comportamento estatístico do usuário, ou seja, quantos arquivos são transferidos a determinado tempo, ou o tamanho de cada arquivo. Considera-se aqui uma modelagem baseada na assunção que a aplicação FTP está gerando um fluxo contínuo de dados; ou seja, está fazendo a transferência de um arquivo muito grande. Os tempos de inatividade da aplicação FTP são desprezados.

É importante atentar para este conceito. Na aplicação VoIP, por exemplo, períodos de inatividade são inerentes à aplicação, pois é o silêncio inerente à fala. No FTP, o intuito é coletar um volume determinado de dados e enviá-lo ao outro *host* de uma só vez, sem pausa.

Quanto à taxa de transmissão, na aplicação FTP ela será determinada, de maneira geral, pela velocidade de processamento dos *hosts*, pela capacidade do *link* e pelo seu estado de utilização. Novamente isto difere das aplicações VoIP e vídeo MPEG; para estas, a taxa de

transmissão é limitada, a priori, pelos *codecs*. Se o *link* proporcionar velocidades maiores que as taxas de geração dos *codecs*, estas aplicações não apresentarão ganhos de performance[11]. Ao FTP interessa a maior velocidade possível.

De posse destas noções, é possível apresentar o código de um gerador de tráfego FTP, extraído de [NS, 2003] e disponível no ANEXO E. O programa foi montado de forma simples e baseado na modelagem apresentada. A seguir, apresenta-se o trecho do programa responsável pela geração do tráfego:

```
Application/FTP instproc start {} {
      [$self agent] send -1
}

# Causes TCP to send no more new data.
Application/FTP instproc stop {} {
      [$self agent] advance 0
      [$self agent] close
}
```

Ao iniciar a aplicação FTP, e portanto sua geração, o programa aciona o agente de transporte TCP, representado pela linha "`[$self agent] send -1`". O comando ou método enviado, "`send -1`", está instruindo o objeto TCP a começar um envio contínuo de segmentos, ininterruptamente, até que se receba o comando para encerrar (o método `stop`, logo abaixo). O TCP agirá como se sempre houvesse dados a transmitir em seu *buffer* de entrada, vindos da aplicação. Caberá ao TCP, pois, o controle de taxa de transmissão, erros e congestionamento.

---

[11] Uma analogia pitoresca seria um automóvel, cuja velocidade máxima é de 120Km/h, trafegando numa autopista cuja velocidade máxima legal é de 80Km/h. Se este limite da autopista for elevado para 180Km/h, o automóvel permanecerá confinado a 120Km/h. Qualquer aumento da velocidade máxima da autopista acima de 120Km/h não representa qualquer ganho para o automóvel.

Para este programa, a geração de tráfego FTP simulada é tão somente o acionamento do serviço de transporte TCP para um número infinito de bytes. O protocolo TCP tratará de dividir este número em segmentos de tamanho MSS (*Maximum Segment Size*) e enviará a uma taxa resultante das janelas de transmissão, dos algoritmos de controle de congestionamento e erro e da capacidade de *link* disponível, que variará conforme a existência e comportamento dos tráfegos concorrentes na rede.

6. CONCLUSÃO

Neste trabalho, analisou-se quatro tipos geração de tráfego por fontes em redes de computadores multi-serviço. Primeiramente, a aplicação Voz sobre IP (VoIP). Na seção seguinte, dedicou-se ao Vídeo Digital comprimido pelo padrão MPEG. A natureza de Auto-Similaridade de algumas fontes de tráfegos de rede foi o tema da seção posterior. O FTP foi o quarto e último tipo de fonte analisada.

Para cada uma delas, partiu-se da caracterização da aplicação envolvida. Então, sugeriu-se uma modelagem, ou mais de uma, que bem representasse o comportamento da aplicação, sob o ponto de vista da rede de computadores. Finalmente, coletou-se um exemplo de código ou programa de computador para gerar o tráfego estudado, em simulação.

Num contexto amplo, o objetivo deste trabalho foi oferecer uma visão da geração de tráfego simulado para os modelos estudados, preferindo uma análise qualitativa e prática. Quando possível, resumir estes modelos num conjunto de parâmetros presentes no simulador de tráfego.

Esta abordagem prática encontrou dificuldades principalmente para a caracterização das fontes MPEG e auto-similar. Aqui, o foco observado na bibliografia consultada é predominantemente matemático, cuja densidade desviar-se-ia do escopo pretendido. Optou-se então por incluir um número de referências para algumas propostas de modelagem destas fontes e códigos baseados nalguma dessas propostas. Em adição, a complexidade dos modelos não permitiu uma síntese apropriada em termos de parâmetros para manipulação em simulação.

Considera-se, com este trabalho, haver sido dada uma contribuição válida para o estudo de fontes de tráfegos em redes de computadores multi-serviço, visando um aspecto

prático das características das fontes de tráfego, modelagens e código, de forma a servir de embrião para confecção de geradores de tráfego sintetizado em simulação.

7. REFERÊNCIAS

[BARROS, 2003] BARROS, Rui Marcos de Oliveira [online]. Disponível em: <http://www.dma.uem.br/~rbarros>. Acesso em 8 de agosto de 2003.

[BERAN, 1995] BERAN, Jan *et al*. Long-range dependence in variable-bit-rate video traffic. **IEEE Transactions on Communications**, vol. 43, n. 234, p. 1566-1579, Feb/Mar/Apr 1995.

[BERGER, 1998] BERGER, Arthur W. *et al*. Standardization of traffic measurements and models for broadband networks: open issues. **Computer Networks and ISDN Systems**, n. 30, p. 1327-1340, Mar. 1998.

[BRITO, 2001] BRITO, Sergio F.; AGUIAR, Mônica V. C. **Características de tráfego das aplicações e da tecnologia do núcleo em redes multiserviços.** Monografia. Universidade Federal de Pernambuco, Recife, 18 jul. 2001.

[BYSTRÖM, 2003] BYSTRÖM, Håkan. [online]. Disponível em: <http://mailman.isi.edu/pipermail/ns-users/2001-October/018453.html>. Acesso em: 10 agosto 2003.

[CARVALHO, 2001] CARVALHO, Rogério Muniz. **Princípios de comunicações**. 3. ed. Vitória ES: Edição do Autor, 2001, p. 180 a 183.

[CERDEIRA, 2001] CERDEIRA, Pablo de Camargo. **Análise de caso: lacração de aparelhos por uso de voz sobre IP**. Trabalho de análise legislativa para uso do escritório Campos Advocacia Empresarial, 17 out. 2001.

[CHANDRA, 1999] CHANDRA, Kavitha; REIBMAN, Amy R. Modeling one- and two-layer variable bit rate video. **IEEE Transactions on Networking**, vol. 7, n. 3, p. 398-413, June 1999.

[CHODOREK, 2000] CHODOREK, Agnieszka; CHODOREK, Robert R. Characterization of the MPEG-2 video traffic generated by DVD applications. In: 1st European Conference on Universal Multiservice Networks, 2000. **Anais…** 2000, p. 62-70.

[CROVELLA, 1997] CROVELLA, Mark E.; BESTAVROS, Azer. Self-similarity in World Wide Web traffic: evidence and possible causes. **IEEE/ACM Transactions on Networking**, vol. 5, n. 6, p. 835-846, Dec. 1997.

[CHUAH, 2002] CHUAH, Chen-Nee; KATZ, Randy H. **Characterizing packet audio streams from Internet multimedia applications**. University of California, Davis e Berkeley, 2002.

[CRUZ, 2003] CRUZ, Tersio Guilherme Souza. **Leis de escala e dimensão fractal em filmes finos:** microscopia de força atômica e voltametria cíclica [online]. Disponível em: <http://www.ifi.unicamp.br/physicae/tersio.pdf>. Acesso em 08 de agosto de 2003.

[DAIGLE, 1986] DAIGLE, John N.; LANFORD, Joseph D. Models for analysis of packet voice communications systems. **IEEE Journal on Selected Areas in Communications,** vol. SAC-4, n. 6, p. 847-855, Sep. 1986.

[DAUT, 1999] DAUT, David G.; YU, Ming. An improved fast algorithm for simulating self-similar traffic with application in ATM networks. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 1999. **Anais…** 1999, p. 14-17.

[ENCARTA, 1999] ENCICLOPÉDIA Microsoft Encarta Encyclopedia Deluxe 2000. Microsoft Corporation 1993-1999.

[GARRETT, 1994] GARRETT, Mark W.; WILLINGER, Walter. Analysis, modeling and generation of self-similar VBR video traffic. In**:** ACM SIGCOMM'94, 1994. **Anais…** 1994, p. 269-280.

[GOODE, 2002] GOODE, Bur. Voice over Internet protocol. **Proceedings of the IEEE**, vol. 90, n. 9, p. 1495-1517, Sep. 2002.

[GORDON, 1995] GORDON, James. Pareto process as a model of self-similar packet traffic. In: IEEE GLOBECOM'95 Global Telecommunications Conference, 1995. **Anais …** vol. 3, Nov. 1995, p. 2232-2236.

[GUIMARÃES, 1999] GUIMARÃES, José Liesse Bollos. **Voz sobre IP** [online]. Disponível em: <http://www.gta.ufrj.br/grad/98_2/liesse/relat.html>. Acesso em: 21 fev. 2003.

[HUANG, 1995a] HUANG, Changcheng *et al*. Fast simulation for self-similar traffic in ATM networks. In: IEEE International Conference on Communications, 1995. **Anais…** 1995, vol. 1, p. 438-444.

[HUANG, 1995b] HUANG, Changcheng *et al*. Modeling and simulation of self-similar variable bit rate compressed video: a unified approach. **ACM SIGCOMM Computer Communication Review**, vol. 25, n. 4, , p. 114-125, Oct. 1995.

[HUANG, 1997] HUANG, Changcheng. **Long range dependent traffic:** modeling, simulation and congestion control. 1997. 125 f. Tese (Doctor of Philosophy) – Department of Systems and Computer Engineering, Ottawa-Carleton Institute for Electrical Engineering.

[JAIN, 1991] JAIN, R. **The art of computer systems perfomance analysis.** John Wiley & Sons, 1991.

[KHAYARI, 2002] KHAYARI, Rachid El Abdouni; SADRE, Ramin. A validation of the Pseudo self-similar traffic model. In: International Conference on Dependable Systems and Networks, 2002. **Anais…**2002, p. 727-734.

[KRISHNA, 2002] KRISHNA, Murali; GADRE, Vikram; DESAI, Uday. Multiplicative multifractal process based modeling of broadband traffic processes: variable bit rate vídeo traffic. In: 2002 International Zürich Seminar on Broadband Communications, 2002, Zürich. **Anais…** Zürich: Feb. 2002, p. 18-1 a 18-6.

[KUROSE, 2001] KUROSE, James F.; ROSS, Keith W. **Computer networking:** a top-down approach featuring the Internet. USA: Addison Wesley Longman, 2001, p. 104-106, 182-258.

[LATHI, 1998] LATHI, B. P. **Modern digital and analog communication systems.** 3. ed. New York: Oxford University Press, 1998, p. 260 a 267.

[LEDESMA, 2000] LEDESMA, Sergio; LIU, Derong. A fast method for generating self-similar network traffic. In: International Conference on Communication Technology Proceedings, 2000. **Anais…** 2000, vol. 1, p. 54-61.

[LELAND, 1994] LELAND, Will E. *et al*. On the self-similar nature of Ethernet traffic (extended version). **IEEE Transactions on Networking**, vol. 2, n. 1, p. 1-15, Feb. 1994.

[LINAWATI, 2002] LINAWATI; MEHRPOUR, H. Self-similar traffic generator: comparison between RMD and SRA methods. 5th IEEE International Conference on High Speed Networks and Multimedia Communications. **Anais…** 2002, p. 37-41, July 2002.

[LIU, 1999a] LIU, Hai; ANSARI, Nirwan; SHI, Yun Q. Modeling VBR video traffic by Markov-modulated self-similar processes. In: IEEE 3rd Workshop on Multimedia Signal Processing, 1999. **Anais…** Sep. 1999, p. 363-368.

[LIU, 1999b] _____. Markov-modulated self-similar processes: MPEG coded video traffic modeler and synthesizer. In: IEEE GLOBECOM'99 Global Telecommunications Conference, 1999. **Anais…** vol. 2, p. 1184-1188, Dec. 1999.

[LIU, 2000] _____. A simple model for MPEG video traffic. In: IEEE International Conference on Multimedia and Expo, 2000. **Anais...** 2000, p. 553-556..

[MA, 1998a] MA, Sheng; JI, Chuanyi. Modeling video traffic in the wavelet domain. In: IEEE INFOCOM'98 17th Annual Joint Conference of the IEEE Computer and Communications Societies, 1998. **Anais…** vol. 1, Mar-Apr. 1998, p. 201-208.

[MA, 1998b] _____. Modeling video traffic using wavelets. In: IEEE International Conference on Communications, 1998. **Anais…** vol. 1, June 1998, p. 559-562.

[MATRAWY, 2002] MATRAWY, Ashraf; LAMBADARIS, Ioannis; HUANG, Changcheng. MPEG4 traffic modeling using the transform expand sample methodology. In: 2002 IEEE 4th International Workshop on Networked Appliances, 2002. **Anais…** Jan. 2002, p. 249-256.

[MATRAWY, 2003] MATRAWY, Ashraf. [online]. Disponível em: <http://www.sce.carleton.ca/~amatrawy/mpeg4>. Acesso em: 10 agosto 2003.

[MELAMED, 1994] MELAMED, B. *et al*. TES-based vídeo source modeling for performance evaluation of integrated networks. **IEEE Transactions on Communications**, vol. 42, n. 10, p. 2773-2777, out. 1994.

[MENA, 2000] MENA, Art; HEIDEMANN, John. An empirical study of Real Audio traffic. In: **Anais IEEE Infocom**, Tel Aviv, Israel, Mar. 2000.

[MENDES Fº., 1996] MENDES FILHO, Jorge Roberto. **Modelos de fonts de tráfego para RDSI-FL** [online]. Disponível em: <http://www.ravel.ufrj.br/publicacoes/rel_9602.pdf>. Acesso em 9 de dezembro de 2003.

[MIRAS, 2002] MIRAS, Dimitrios. **A Survey on Network QoS Needs of Advanced Internet Applications** [online]**.** Disponível em: <http://www.cs.ucl.ac.uk/staff/D.Miras/>. Acesso em: 01 dezembro 2003.

[NS, 2003] **NETWORK SIMULATOR** (ns) [online]. Disponível em: <http://www.isi.edu/nsnam>. Acesso em: 20 maio 2003.

[NUNES, 2002] NUNES, Alessandro do Valle. **Voz sobre IP** [online]. Monografia apresentada como requerimento da disciplina Teleprocessamento e Redes, do curso de Bacharelado em Informática pela Universidade Católica do Salvador. Disponível em: <http://www.logicengenharia.com.br/mcamara/ALUNOS/voip.html>. Acesso em: 15 maio 2003.

[PAXSON, 1997] PAXSON, Vern. Fast, approximate synthesis of fractional Gaussian noise for generating self-similar network traffic. **ACM SIGCOMM Computer Communication Review**, vol. 27, n. 5, p. 5-18, Jan. 1997.

[PORTNOI, 2002] PORTNOI, Marcos; ARAÚJO, Rafael G. B. Network Simulator – visão geral da ferramenta de simulação de redes. **SEPA - Seminário Estudantil de Produção Acadêmica.** Salvador, ano VI, n. 6, v. 6, p. 173-181, 2002.

[PRASAD, 1996] PRASAD, Anand R.; STAVROV, Borut; SCHOUTE, Frits C. Generation and testing of self-similar traffic in ATM networks. IEEE International Conference on Personal Wireless Communications, 1996. **Anais…** Feb. 1996, p. 200-205.

[SAHINOGLU, 1999] SAHINOGLU, Zafer; TEKINAY, Sirin. On multimedia networks: self-similar traffic and network performance. **IEEE Communications Magazine**, Jan. 1999.

[SANTOS, 2003] SANTOS, Celso A. Saibel. **Aplicações Multimídia.** Material didático do Curso de Graduação em Engenharia Elétrica da UNIFACS. Salvador, BA, 2003.

[SCHULER, 1996] SCHULER, Christian. Disponível em: <http://ita.ee.lbl.gov/html/contrib/fft_fgn_c.html>. Acesso em: 06 dezembro 2003.

[SCHWARTZ, 1996] SCHWARTZ, Mischa. **Broadband Integrated Networks.** 1. ed. Simon & Schuster Adult Publishing Group, 1996, p. 21 a 113.

[SHIN, 1995] SHIN, Jae-Jin; KIM, Jae-Kyoon; SUNG, Dan Keun. Characterization of variable bit rate video traffic as vector autoregressive processes. In: IEEE GLOBECOM'95 Global Telecommunications Conference, 1995. **Anais …** vol. 2, Nov. 1995, p. 1427-1431,

[SILVA, 1999] SILVA, Fernanda M. A. **Multiplexação estatística de fontes heterogêneas em ATM** [online]. Disponível em: <http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana99/fernandamas/fernandamas.html>. Acesso em: 08 agosto 2003.

[TANENBAUM, 1996] TANENBAUM, Andrew S. **Computer networks.** 3. ed. New Jersey: Prentice Hall, 1996, 813 p.

[TAQQU, 1997] TAQQU, Murad S.; WILLINGER, Walter; SHERMAN, Robert. Proof of a fundamental result in self-similar traffic modeling. **Computer Communications Review**, vol. 27, n. 2, p. 5-23, abr. 1997.

[TECHTARGET, 2003] TECHTARGET Network. **Whatis?com** [online]. Disponível em: <http://www.whatis.com>. Acesso em: 10 dez 2003.

[TELEFONIA IP, 2002] TELEFONIA IP (VoIP) [online]. Disponível em: <http://www.gta.ufrj.br/~rezende/cursos/eel879/trabalhos/voip/>. Acesso em: 21 fev. 2003.

[TELEMAR, 2001] TELEMAR oferece tecnologia ISDN com o produto DVI [online]. **Jornal do Commercio**, Recife, 07 março 2001. Disponível em: <http://www2.uol.com.br/JC/_2001/0703/if0703_3.htm>. Acesso em 20 dezembro 2003.

[WAKAMIYA, 2000] WAKAMIYA, N.; MURATA, M.; MIYAHARA, H. On video coding algorithms with application level QoS guarantees. **Computer Communications**, n. 23, p. 1459-1469, 2000.

[WANG, 1997] WANG, Xinyu; JUNG, Sounhwan; MEDITCH, James S. VBR broadcast video traffic modeling – a wavelet decomposition approach. In: IEEE GLOBECOM'97 Global Telecommunications Conference, 1997. **Anais…** vol. 2, Nov. 1997, p. 1052-1056.

[XU, 1999] XU, W.; QURESHI, A. G. Adaptative Linear Prediction of MPEG video traffic. In: Fifth International Symposium on Signal Processing and its Applications – ISSPA'99, 1999, Australia. **Anais…** Austrália: 1999.

[XUYONG, 2000] XUYONG, Dongying; LIUYAN, Zhanglicui. Modeling and the simulation study of MPEG video traffic. In: 3rd World Congress on Intelligent Control and Automation, 2000. **Anais…** vol. 3, p. 2158-2162, June 2000.

ANEXO A – Modelagem de Fluxos

Descreve-se a seguir alguns modelos estocásticos para caracterização de fontes de tráfego de dados, voz e vídeo [SILVA, 1999].

MODELO DE POISSON

Analiticamente o mais simples, o modelo baseado em processo de Poisson é bastante utilizado no dimensionamento de redes. Apresenta limitações, como por exemplo a falha em capturar a correlação das chegadas de forma satisfatória e também a explosividade (*burstiness*) do tráfego, fatores estes que afetam o desempenho em filas. Essas limitações não indicam o modelo de Poisson para descrever fluxos agregados e fluxos que apresentam dependência de longo alcance (LRD – *Long Range Dependency*).

MODELO ON/OFF EXPONENCIAL

Fontes de tráfego On/Off Exponenciais alternam períodos ativos de emissão (períodos ON) e períodos inativos (períodos OFF). Durante o período ativo ON, os pacotes ou dados são gerados a uma taxa constante (caracterizando uma rajada ou *burst*). No período inativo ou OFF, nenhum dado ou pacote é gerado. Estes períodos são caracterizados por uma média ou tempo de interchegada modelados por uma distribuição exponencial. O modelo On/Off Exponencial é bastante usado para modelar tráfego de voz, e é capaz de reproduzir correlações e explosividade de um tráfego real.

PROCESSOS MARKOVIANOS

Pode-se citar três modelos baseados no processo de Markov: MMPP (*Markov Modulated Poisson Process*, ou Processo de Poisson Modulado por Markov); IPP (*Interrupted Poisson Process*, ou Processo de Poisson Interrompido) e IBP (*Interrupted*

Bernoulli Process, ou Processo de Bernoulli Interrompido). Boa captura da correlação de chegadas e explosividade do tráfego, características não abrangidas pelo modelo de Poisson, são modeladas pelo MMPP. Um processo MMPP foi usado para modelar as chegadas agregadas de diversas fontes de voz e dados, e um MMPP multi-estado foi usado para modelar superposição de tráfego multimídia. O IPP e IBP são indicados para tráfego de voz.

MODELO DE FLUXOS (*STOCHASTIC FLUID FLOW MODEL)*

Os modelos baseados em fluxo formam uma classe onde as células ATM são geradas continuamente, assemelhando-se a um fluxo ou um fluido, em vez de instantes individuais de tempo.

MODELO AUTO-SIMILAR (*SELF-SIMILAR*)

Visualmente, a auto-similaridade de um tráfego pode ser entendida como uma invariância em relação à escala de tempo. Se se construírem gráficos do tipo "pacotes/unidade de tempo" versus "unidade de tempo", poder-se-á verificar que o tráfego parece o mesmo na escala de horas, minutos, segundos, milissegundos. Vários autores defendem este tipo de modelagem para diversos tipos de tráfego. O aumento de utilização numa rede Ethernet, por exemplo, intensifica a explosividade do tráfego. Isso está em discordância com os modelos de tráfego baseados em Poisson e MMPP, pois em tais modelos, o tráfego agregado torna-se mais suave (menos explosivo) com o aumento da quantidade de fontes [LELAND, 1994]. O modelo auto-similar tende a descrever mais realisticamente tal caso, bem como tráfego de vídeo VBR e várias fontes On/Off agregadas.

MODELO ON/OFF PARETO

A diferença entre o modelo On/Off Pareto e o On/Off Exponencial é que, no primeiro, os períodos On/Off são estatisticamente arranjados segundo a distribuição de Pareto

(enquanto que o último o faz pela distribuição exponencial). Estudos descritos em [TAQQU, 1997] indicam que a superposição de várias fontes On/Off Pareto (com estrita alternância entre os períodos e igualmente distribuídas, chamadas de "*packet trains*" ou trens de pacote) resultam em tráfego agregado auto-similar.

ANEXO B – Exemplo de Código para Fonte Exponencial On/Off

Este código foi extraído do simulador *ns*, disponível em [NS, 2003], arquivo `expoo.cc`.

Incluiu-se após o código do arquivo `rng.h`, para melhor compreensão da geração do número randômico de distribuição exponencial. Recomenda-se consultar também os códigos `random.cc`, `random.h`, `rng.cc`, `trafgen.cc` e `trafgen.h` em [NS, 2003].

```
/* -*-      Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t -*-
*/
/*
 * Copyright (c) Xerox Corporation 1997. 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 Xerox is
 * acknowledged in all documentation pertaining to any such copy or
derivative
 * work. Xerox grants no other licenses expressed or implied. The Xerox
trade
 * name should not be used in any advertising without its written
permission.
 *
 * XEROX CORPORATION 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.
 */

#ifndef lint
static const char rcsid[] =
    "@(#) $Header: /nfs/jade/vint/CVSROOT/ns-2/tools/expoo.cc,v 1.12
2000/10/13 19:44:15 debo Exp $ (Xerox)";
#endif

#include <stdlib.h>

#include "random.h"
#include "trafgen.h"
#include "ranvar.h"


/* implement an on/off source with exponentially distributed on and
 * off times.  parameterized by average burst time, average idle time,
 * burst rate and packet size.
 */
```

```
class EXPOO_Traffic : public TrafficGenerator {
 public:
      EXPOO_Traffic();
      virtual double next_interval(int&);
        virtual void timeout();
      // Added by Debojyoti Dutta October 12th 2000
      int command(int argc, const char*const* argv);
 protected:
      void init();
      double ontime_;   /* average length of burst (sec) */
      double offtime_;  /* average length of idle time (sec) */
      double rate_;     /* send rate during on time (bps) */
      double interval_; /* packet inter-arrival time during burst (sec) */
      unsigned int rem_; /* number of packets left in current burst */

      /* new stuff using RandomVariable */
      ExponentialRandomVariable burstlen_;
      ExponentialRandomVariable Offtime_;

};


static class EXPTrafficClass : public TclClass {
 public:
      EXPTrafficClass() : TclClass("Application/Traffic/Exponential") {}
      TclObject* create(int, const char*const*) {
            return (new EXPOO_Traffic());
      }
} class_expoo_traffic;

// Added by Debojyoti Dutta October 12th 2000
// This is a new command that allows us to use
// our own RNG object for random number generation
// when generating application traffic

int EXPOO_Traffic::command(int argc, const char*const* argv){

        if(argc==3){
                if (strcmp(argv[1], "use-rng") == 0) {
                        burstlen_.seed((char *)argv[2]);
                        Offtime_.seed((char *)argv[2]);
                        return (TCL_OK);
                }
        }
        return Application::command(argc,argv);
}


EXPOO_Traffic::EXPOO_Traffic() : burstlen_(0.0), Offtime_(0.0)
{
      bind_time("burst_time_", &ontime_);
      bind_time("idle_time_", Offtime_.avgp());
      bind_bw("rate_", &rate_);
      bind("packetSize_", &size_);
}

void EXPOO_Traffic::init()
{
        /* compute inter-packet interval during bursts based on
       * packet size and burst rate.  then compute average number
```

```
	 * of packets in a burst.
	 */
	interval_ = (double)(size_ << 3)/(double)rate_;
	burstlen_.setavg(ontime_/interval_);
	rem_ = 0;
	if (agent_)
		agent_->set_pkttype(PT_EXP);
}

double EXPOO_Traffic::next_interval(int& size)
{
	double t = interval_;

	if (rem_ == 0) {
		/* compute number of packets in next burst */
		rem_ = int(burstlen_.value() + .5);
		/* make sure we got at least 1 */
		if (rem_ == 0)
			rem_ = 1;
		/* start of an idle period, compute idle time */
		t += Offtime_.value();
	}
	rem_--;

	size = size_;
	return(t);
}

void EXPOO_Traffic::timeout()
{
	if (! running_)
		return;

	/* send a packet */
	// The test tcl/ex/test-rcvr.tcl relies on the "NEW_BURST" flag being
	// set at the start of any exponential burst ("talkspurt").
	if (nextPkttime_ != interval_ || nextPkttime_ == -1)
		agent_->sendmsg(size_, "NEW_BURST");
	else
		agent_->sendmsg(size_);
	/* figure out when to send the next one */
	nextPkttime_ = next_interval(size_);
	/* schedule it */
	if (nextPkttime_ > 0)
		timer_.resched(nextPkttime_);
}
```

Código do arquivo `rng.h`.

```
/* -*-	Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t -*-
*/
/*
 * Copyright (c) 1997 Regents of the University of California.
 * All rights reserved.
```

```
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    This product includes software developed by the Computer Systems
 *    Engineering Group at Lawrence Berkeley Laboratory.
 * 4. Neither the name of the University nor of the Laboratory may be used
 *    to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 *   FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 *  HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * "@(#) $Header: /nfs/jade/vint/CVSROOT/ns-2/tools/rng.h,v 1.22 2002/01/23
21:38:23 buchheim Exp $ (LBL)";
 */

/*************************************************************************\
*
* File: RngStream.h for multiple streams of Random Numbers
* Language: C++
* Copyright: Pierre L'Ecuyer, University of Montreal
* Notice: This code can be used freely for personnal, academic,
* or non-commercial purposes. For commercial purposes,
* please contact P. L'Ecuyer at: lecuyer@iro.umontreal.ca
* Date: 14 August 2001
*
* Incorporated into rng.h and modified to maintain backward
* compatibility with ns-2.1b8.  Users can use their current scripts
* unmodified with the new RNG.  To get the same results as with the
* previous RNG, define OLD_RNG when compiling (e.g., make -D OLD_RNG).
* - Michele Weigle, University of North Carolina (mcweigle@cs.unc.edu)
* October 10, 2001
*************************************************************************/

/* new random number generator */

#ifndef _rng_h_
#define _rng_h_
```

```
#ifdef rng_stand_alone
#define stand_alone
#define rng_test
#endif

#include <math.h>
#include <stdlib.h>               // for atoi

#ifndef stand_alone
#include "config.h"
#endif   /* stand_alone */

#ifndef MAXINT
#define     MAXINT      2147483647  // XX [for now]
#endif

#ifdef OLD_RNG
/*
 * RNGImplementation is internal---do not use it, use RNG.
 */
class RNGImplementation {
public:
      RNGImplementation(long seed = 1L) {
            seed_ = seed;
      };
      void set_seed(long seed) { seed_ = seed; }
      long seed() { return seed_; }
      long next();                  // return the next one
      double next_double();
private:
      long seed_;
};
#endif /* OLD_RNG */

/*
 * Use class RNG in real programs.
 */
class RNG
#ifndef stand_alone
      : public TclObject
#endif  /* stand_alone */
{

public:
      enum     RNGSources     {     RAW_SEED_SOURCE,     PREDEF_SEED_SOURCE,
HEURISTIC_SEED_SOURCE };

#ifdef OLD_RNG
      RNG() : stream_(1L) {};
      inline int seed() { return stream_.seed(); }
#else
      RNG(const char* name = "");
      RNG(long seed);
      void init();
      long seed();
      void set_seed (long seed);
      long next();
      double next_double();
#endif /* OLD_RNG */
```

```
      RNG(RNGSources source, int seed = 1) { set_seed(source, seed); };
      void set_seed(RNGSources source, int seed = 1);
      inline static RNG* defaultrng() { return (default_); }

#ifndef OLD_RNG
      /*
       * Added for new RNG
       */
      static void set_package_seed (const unsigned long seed[6]);
      /*
        Sets the initial seed s 0 of the package to the six integers in the
        vector seed. The first 3 integers in the seed must all be less than
        m 1 = 4294967087, and not all 0; and the last 3 integers must all
be
        less than m 2 = 4294944443, and not all 0. If this method is not
        called, the default initial seed is (12345, 12345, 12345, 12345,
        12345, 12345).
      */

      void reset_start_stream ();
      /*
        Reinitializes the stream to its initial state: C g and B g are set
        to I g
      */

      void reset_start_substream ();
      /*
        Reinitializes the stream to the beginning of its current substream:
        C g is set to B g.
      */

      void reset_next_substream ();
      /*
        Reinitializes the stream to the beginning of its next substream: N
g
        is computed, and C g and B g are set to N g .
      */

      void set_antithetic (bool a);
      /*
        If a = true, the stream will start generating antithetic variates,
        i.e., 1 - U instead of U,until this method is called again with a =
        false.
      */

      void increased_precis (bool incp);
      /*
        After calling this method with incp = true, each call to the
        generator (direct or indirect) for this stream will return a
uniform
        random number with more bits of resolution (53 bits if machine
        follows IEEE 754 standard) instead of 32 bits, and will advance the
        state of the stream by 2 steps instead of 1. More precisely, if s
is
        a stream of the class RngStream, in the non- antithetic case, the
        instruction ``u = s.RandU01()'' will be equivalent to ``u =
        (s.RandU01() + s.RandU01() * fact) % 1.0'' where the constant fact
        is equal to 2 -24 . This also applies when calling RandU01
        indirectly (e.g., via RandInt, etc.). By default, or if this method
        is called again with incp = false, each to call RandU01 for this
        stream advances the state by 1 step and returns a number with 32
```

```
        bits of resolution.
      */

      void set_seed (const unsigned long seed[6]);
      /*
        Sets the initial seed I g of the stream to the vector seed. The
        vector seed should contain valid seed values as described in
        SetPackageSeed. The state of the stream is then reset to this
        initial seed. The states and seeds of the other streams are not
        modified. As a result, after calling this method, the initial seeds
        of the streams are no longer spaced Z values apart. We discourage
        the use of this method; proper use of the Reset* methods is
        preferable.
      */

      void advance_state (long e, long c);
      /*
        Advances the state by n steps (see below for the meaning of n),
        without modifying the states of other streams or the values of B g
        and I g in the current object. If e > 0, then n =2 e + c;if e < 0,
        then n = -2 -e + c;and if e = 0,then n = c. Note:c is allowed to
        take negative values.  We discourage the use of this method.
      */

      void get_state (unsigned long seed[6]) const;
      /*
        Returns in seed[0..5] the current state C g of this stream. This is
        convenient if we want to save the state for subsequent use.
      */

      void write_state () const;
      /*
        Writes (to standard output) the current state C g of this stream.
      */

      void write_state_full () const;
      /*
        Writes (to standard output) the value of all the internal variables
        of this stream: name, anti, incPrec, Ig, Bg, Cg.
      */

      double rand_u01 ();
      /*
        Normally, returns a (pseudo)random number from the uniform
        distribution over the interval (0, 1), after advancing the state by
        one step. The returned number has 32 bits of precision in the sense
        that it is always a multiple of 1/(2 32 - 208). However, if
        IncreasedPrecis(true) has been called for this stream, the state is
        advanced by two steps and the returned number has 53 bits of
        precision.
      */

      long rand_int (long i, long j);
      /*
        Returns a (pseudo)random number from the discrete uniform
distribution
        over the integers {i, i +1,...,j}. Makes one call to RandU01.
      */
#endif /* !OLD_RNG */

#ifndef stand_alone
```

```
      int command(int argc, const char*const* argv);
#endif  /* stand_alone */

      // These are primitive but maybe useful.
      inline int uniform_positive_int() {  // range [0, MAXINT]
#ifdef OLD_RNG
            return (int)(stream_.next());
#else
            return (int)(next());
#endif /* OLD_RNG */
      }
      inline double uniform_double() { // range [0.0, 1.0)
#ifdef OLD_RNG
            return stream_.next_double();
#else
            return next_double();
#endif /* OLD_RNG */
      }

      // these are for backwards compatibility
      // don't use them in new code
      inline int random() { return uniform_positive_int(); }
      inline double uniform() {return uniform_double();}

      // these are probably what you want to use
      inline int uniform(int k)
      { return (uniform_positive_int() % (unsigned)k); }
      inline double uniform(double r)
      { return (r * uniform());}
      inline double uniform(double a, double b)
      { return (a + uniform(b - a)); }
      inline double exponential()
      { return (-log(uniform())); }
      inline double exponential(double r)
      { return (r * exponential());}
      // See "Wide-area traffic: the failure of poisson modeling", Vern
      // Paxson and Sally Floyd, IEEE/ACM Transaction on Networking, 3(3),
      // pp. 226-244, June 1995, on characteristics of counting processes
      // with Pareto interarrivals.
      inline double pareto(double scale, double shape) {
            // When 1 < shape < 2, its mean is scale**shape, its
            // variance is infinite.
            return (scale * (1.0/pow(uniform(), 1.0/shape)));
      }
        inline double paretoII(double scale, double shape) {
            return (scale * ((1.0/pow(uniform(), 1.0/shape)) - 1));
      }
      double normal(double avg, double std);
      inline double lognormal(double avg, double std) {
            return (exp (normal(avg, std)));
      }

protected:   // need to be public?
#ifdef OLD_RNG
      RNGImplementation stream_;
#else
      double Cg_[6], Bg_[6], Ig_[6];
      /*
        Vectors to store the current seed, the beginning of the current
block
        (substream) and the beginning of the current stream.
```

```
      */

      bool anti_, inc_prec_;
      /*
        Variables to indicate whether to generate antithetic or increased
        precision random numbers.
      */

      char name_[100];
      /*
        String to store the optional name of the current RngStream object.
      */

      static double next_seed_[6];
      /*
        Static vector to store the beginning state of the next RngStream to
        be created (instantiated).
      */

      double U01 ();
      /*
        The backbone uniform random number generator.
      */

      double U01d ();
      /*
        The backbone uniform random number generator with increased
        precision.
      */
#endif /* OLD_RNG */
      static RNG* default_;
};

/*
 * Create an instance of this class to test RNGImplementation.
 * Do .verbose() for even more.
 */
#ifdef rng_test
class RNGTest {
public:
      RNGTest();
      void verbose();
      void first_n(RNG::RNGSources source, long seed, int n);
};
#endif /* rng_test */

#endif /* _rng_h_ */
```

ANEXO C – Exemplo de Código para Fonte de Tráfego VBR ou MPEG

Dois exemplos são fornecidos. A seguir, o arquivo de autoria de [MATRAWY, 2003], denominado `mpeg4_traffic.cc`.

```
/* -*-      Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t -*-
*t
/*
 * Copyright (c) Xerox Corporation 1997. 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 Xerox is
 * acknowledged in all documentation pertaining to any such copy or
derivative
 * work. Xerox grants no other licenses expressed or implied. The Xerox
trade
 * name should not be used in any advertising without its written
permission.
 *
 * XEROX CORPORATION 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.
 */

/*
 * A source for MPEG4 traffic.
 * Produces traffic that has the same marginal distribution and
 * autocorrelation function of a specified mpeg4 trace.
 * A TES model is developed first to get the statistics and then
 * the statistics are used to generate the traffic
 */

#include <stdlib.h>
#include "random.h"
#include "trafgen.h"
#include "ranvar.h"

class Frame {
 public:
   Frame(const char*, const char*);
   void Get_Frame(double &,int &);   // get the next frame size
 protected:
   RNG* rng_;
   int numEntry_inv_;              // number of entries in the innovation
CDF table
   int numEntry_hist_;             // number of entries in the frame size
CDF table
```

```
   int maxEntry_;                  // size of the CDF table (mem
allocation)
   CDFentry* table_hist_;          // CDF table of frame sizes (val_, cdf_)
   CDFentry* table_inv_;           // CDF table of innovation (val_, cdf_)
   double value(double, CDFentry*, int );
   double interpolate(double x, double x1, double y1, double x2, double
y2);
   int lookup(double u,CDFentry*,int);
   void loadCDF(const char* filename, CDFentry*& , int &);
};

Frame::Frame(const char* file_1, const char* file_2)  :  maxEntry_(50),
table_inv_(0), table_hist_(0)
{
   rng_ = RNG::defaultrng();
   loadCDF (file_1,table_hist_,numEntry_hist_);
   loadCDF (file_2,table_inv_,numEntry_inv_);
}

void Frame::Get_Frame (double &U, int &size)
{
  size  = int(value(U, table_hist_,numEntry_hist_));
  double rnd = rng_->uniform(0,1);
  double w = value (rnd,table_inv_,numEntry_inv_);
  U = U + w;
  U = U - floor(U);
  //smoothing using eata = 0.5
  if (U<0.5) U = U/0.5;
  else U = (1-U)/0.5;
}

double Frame::value(double u, CDFentry *table_, int numEntry_)
{
   if (numEntry_ <= 0) return 0;
      int mid = lookup(u,table_,numEntry_);
            return interpolate(u, table_[mid-1].cdf_, table_[mid-1].val_,
                           table_[mid].cdf_, table_[mid].val_);
}

double Frame::interpolate(double x, double x1, double y1, double x2, double
y2)
{
      return (y1 + (x - x1) * (y2 - y1) / (x2 - x1));
}

int Frame::lookup(double u, CDFentry* table_, int numEntry_)
{
      // always return an index whose value is >= u
      int lo, hi, mid;
      if (u <= table_[0].cdf_)
            return 0;
      for (lo=1, hi=numEntry_-1;  lo < hi; ) {
            mid = (lo + hi) / 2;
            if (u > table_[mid].cdf_)
                  lo = mid + 1;
            else hi = mid;
      }
      return lo;
}
```

```
void  Frame::loadCDF(const char* filename, CDFentry*& table_, int &
numEntry_)
{
      FILE* fp;
      char line[256];
      CDFentry* e;
      fp = fopen(filename, "r");
      if (fp == 0)
            return ;
      if (table_ == 0)
            table_ = new CDFentry[maxEntry_];
      for (numEntry_=0;  fgets(line, 256, fp);  numEntry_++) {
            e = &table_[numEntry_];
            sscanf(line, "%lf %lf", &e->val_, &e->cdf_);
      }
      return ;
}

//////////////////////////////////////////////////////////////////////////
//////////////////////


class VIDEO_Traffic : public TrafficGenerator {
 public:
      VIDEO_Traffic();
      virtual double next_interval(int&);
        double u0;
        double rateFactor_;
protected:
        RNG* rng_;
        const char* I_File_1;
        const char* I_File_2;
        const char* P_File_1;
        const char* P_File_2;
        const char* B_File_1;
        const char* B_File_2;
        char prev_frame_type, next_frame_type;
        double inter_frame_interval_ ;
      int GOP_count;
      virtual void start();
      virtual void timeout();
        double Ui, Up, Ub;              // random seeds for the three frame
types
        Frame *i;
      Frame *p;
      Frame *b;
};


static class VIDEOTrafficClass : public TclClass {
 public:
      VIDEOTrafficClass() : TclClass("Application/Traffic/MPEG4") {}
      TclObject* create(int, const char*const*) {
            return (new VIDEO_Traffic());
      }
} class_video_traffic;

VIDEO_Traffic::VIDEO_Traffic()
{
        bind("initialSeed_",&u0);
        bind("rateFactor_",&rateFactor_);
```

```
      I_File_1 = "./video_model/Imodel_hist_1";
      I_File_2 = "./video_model/Imodel_inv_1";
      P_File_1 = "./video_model/Pmodel_hist_1";
      P_File_2 = "./video_model/Pmodel_inv_1";
      B_File_1 = "./video_model/Bmodel_hist_1";
      B_File_2 = "./video_model/Bmodel_inv_1";
        rng_ = RNG::defaultrng();
}

void VIDEO_Traffic::start()
{
        Ui = u0;
        size_ = 0;
      next_frame_type = 'I';
      prev_frame_type = ' ';
      GOP_count = 3;                        //first GOP is 10 frames only
      inter_frame_interval_ = 1.0/30.0;   // 30 frame/sec
      i = new Frame (I_File_1,I_File_2);
      p = new Frame (P_File_1,P_File_2);
      b = new Frame (B_File_1,B_File_2);
      if (agent_) agent_->set_pkttype(PT_VIDEO);
        running_ = 1;
        timeout();
}

void VIDEO_Traffic::timeout()
{
        double frame_in_bytes;
        if (! running_)
                return;
        /* figure out when to send the next one */
          nextPkttime_ = next_interval(size_);
        /* send a packet */
        frame_in_bytes = rateFactor_*size_/8 ;
        agent_->sendmsg(frame_in_bytes);
      /* schedule it */
        if (nextPkttime_ > 0)
                timer_.resched(nextPkttime_);
        else
                running_ = 0;
}

double VIDEO_Traffic::next_interval(int& size)
{
      if (next_frame_type == 'I'){
              i->Get_Frame(Ui, size);
            Up = Ui;
            Ub = Ui;
            if (prev_frame_type == 'B')
                    next_frame_type = 'B';
              else  next_frame_type = 'P';
              prev_frame_type = 'I';
      } // end if == I
      else if (next_frame_type == 'P'){
            p->Get_Frame (Up, size);
            next_frame_type = 'B';
              prev_frame_type = 'P';
      } // end if == P
      else if (next_frame_type == 'B'){
            b->Get_Frame (Ub, size);
            if (prev_frame_type != 'B') next_frame_type = 'B';
```

```
                              else if (GOP_count == 12)
                                    {next_frame_type = 'I'; GOP_count = 0;}
                                          else next_frame_type = 'P';
              prev_frame_type = 'B';
      } // end if == B
      GOP_count++ ;
      return(inter_frame_interval_);
}
```

O exemplo seguinte é de autoria de [BYSTRÖM, 2003], arquivo vbr_traffic.cc.

```
/* -*-      Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t -*-
*/
/*
 * Copyright (c) Xerox Corporation 1997. 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 Xerox is
 * acknowledged in all documentation pertaining to any such copy or
derivative
 * work. Xerox grants no other licenses expressed or implied. The Xerox
trade
 * name should not be used in any advertising without its written
permission.
 *
 * XEROX CORPORATION 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.
 */

#include <stdlib.h>

#include "random.h"
#include "trafgen.h"
#include "ranvar.h"

/*
 * The VBR_Traffic class represents a variable bit rate traffic source.
 * Parameterized by rate, deviation of rate in percent, time periods of no
 * burst, time period of burst, number of rate changes during burst,
 * deviation of time periods in percent and constant rate variation or not.
 */
class VBR_Traffic : public TrafficGenerator {
 public:
      VBR_Traffic();
      virtual double next_interval(int&);
```

```
      //HACK so that udp agent knows interpacket arrival time within a
burst
      inline double interval() { return (interval_); }
 protected:
      virtual void start();
      void init();
      double rate_;       /* Send rate during on time (bps). */
      double rate_dev_;   /* Size of random variation on rate. */
      double rate_time_;  /* Mean time of ordinary rate period. */
      double burst_time_; /* Mean time of burst rate period. */
      double n_o_changes_;/* Number of changes during burst time. */
      double time_dev_;   /* Deviation from time periods. */
      int constant_;      /* Perform constant bit rate variation?
                           * NOTICE: burst_time_ controls time between
                           * rate variation. */
      double maxrate_;    /* Maximum rate of burst / constant variation. */
      int maxpkts_;
      int seqno_;

private:
      double next_change; /* Time for next rate change. */
      double next_burst;  /* Time for next burst. */
      double next_rate;   /* Time for next ordinare rate. */
      double curr_rate;
      double interval_;   /* Packet inter-arrival time during burst (sec)
*/
};


static class VBRTrafficClass : public TclClass {
 public:
      VBRTrafficClass() : TclClass("Application/Traffic/VBR") {}
      TclObject* create(int, const char*const*) {
            return (new VBR_Traffic());
      }
} class_vbr_traffic;

VBR_Traffic::VBR_Traffic() : seqno_(0) {
      /* See comments in class definition above. */
      bind_bw("rate_", &rate_);               //default 448Kb
      bind("rate_dev_", &rate_dev_);          //default 0.25
      bind_time("rate_time_", &rate_time_);   //default 2.0
      bind_time("burst_time_", &burst_time_); //default 1.0
      bind("n_o_changes_", &n_o_changes_);    //default 10
      bind("time_dev_", &time_dev_);          //default 0.5
      bind_bool("constant_", &constant_);     //default false
      bind_bw("maxrate_", &maxrate_);         //default 648Kb
      bind("packetSize_", &size_);            //default 210
      bind("maxpkts_", &maxpkts_);            //default 268435456
}

void VBR_Traffic::init() {
        double now = Scheduler::instance().clock();
      //Start with burst.
      next_change = now;
      next_burst = now;
      next_rate = now + burst_time_ * (1 - Random::
                              uniform(-time_dev_, time_dev_));
      curr_rate = rate_;

        // compute inter-packet interval
```

```
	interval_ = (double)(size_ << 3)/(double)rate_;
	if (agent_) {
		//Use same packet type as for CBR.
		agent_->set_pkttype(PT_CBR);
	}
}

void VBR_Traffic::start()
{
	  init();
	  running_ = 1;
	  timeout();
}

/*
 * Returns the time until the next packet is created (and also sets the
size
 * in bytes of the next packet if it has changed in the tcl script).
 */
double VBR_Traffic::next_interval(int& size) {
	  double now = Scheduler::instance().clock();

	if (next_change <= now) {
		// Randomize rate and deviation from desired time.
		double rand_time = Random::uniform(-time_dev_, time_dev_);
		double rand = Random::uniform(-(rate_dev_), rate_dev_);
		curr_rate = curr_rate + (maxrate_ - rate_) * rand;

		if (constant_) {
			// Set time for next rate change.
			next_change = now + burst_time_ * (1 - rand_time);
		} else if (next_burst <= next_rate) {
			// Set time for next rate change within burst.
			next_burst = now + burst_time_ * (1 - rand_time) /
				n_o_changes_;
		} else {
			/* End of burst period. Set times for next burst
			 * period. */
			next_change = now + rate_time_ * (1 - rand_time);
			next_burst = next_change;
			rand_time = Random::uniform(-time_dev_, time_dev_);
			next_rate = next_change + burst_time_ *
				(1 - rand_time);
			curr_rate = rate_;
		}

		// Limit the result.
		if (curr_rate > maxrate_)
			curr_rate = maxrate_;
		if (curr_rate < rate_)
			curr_rate = rate_;

	}

	interval_ = (double)(size_ << 3)/(double)curr_rate;
	double t = interval_;

	size = size_;
	if (++seqno_ < maxpkts_)
		return(t);
	else
```

```
            return(-1);
}
```

ANEXO D – Exemplo de Código para Gerador de Tráfego Auto-Similar

O código abaixo, em C, escrito por [SCHULER, 1996], é uma adaptação do código original em linguagem S (da AT&T, para ambientes estatísticos) confeccionado por [PAXSON, 1997]. Trata-se de um gerador de ruído gaussiano fracional (FGN – *Fractional Gaussian Noise*) baseado em algoritmo FFT (Transformada Rápida de Fourier).

```
/*  $Header:  /net/nthp/u1/chs/cvsroot/fft_fgn_c/fft_fgn.c,v  1.4  1996/01/05
14:11:29 chs Exp $ */

/***************************************************************************
*
* file:       fft_fgn.c
* author:     Christian Schuler
*             Research Institute for Open Communication Systems
*             GMD FOKUS, Hardenbergplatz 2, D-10623 Berlin, Germany
*             Phone : ++49 / (0)30 / 254 99 - 295
*             email : schuler@fokus.gmd.de
* date:       25.8.95

Copyright (c) 1995 The Regents of the University of California.
All rights reserved.

This code is derived from software contributed to Berkeley by
Vern Paxson.

The United States Government has rights in this work pursuant
to contract no. DE-AC03-76SF00098 between the United States
Department of Energy and the University of California.

Redistribution and use in source and binary forms are permitted
provided that: (1) source distributions retain this entire
copyright notice and comment, and (2) distributions including
binaries display the following acknowledgement:  ``This product
includes software developed by the University of California,
Berkeley and its contributors'' in the documentation or other
materials provided with the distribution and in all advertising
materials mentioning features or use of this software.  Neither the
name of the University nor the names of its contributors may be
used to endorse or promote products derived from this software
without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.

-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
This basically says "do whatever you please with this software except
remove this notice or take advantage of the University's (or the author's)
name".
```

```
In particular, you are free to redistribute the software, sell it,
modify it, etc., provided you abide by the terms above.

*
* description:
*    Generation of Fractional Gaussian Noise by the FFT-Algorithm proposed
by
*   Vern Paxson, "Fast Approximation of Self-Similar Network Traffic",
*   Berkeley 1995
*   Info:
*   "http://town.hall.org/Archives/pub/ITA/html/contrib/fft-fgn.html"
*
* known errors:
*  The transformation of the inverse fourier transform to an integer output
*   will probably change the statistical properties of the output sample
path.
*
*
*  modified:
*    chs 4.9.95   different output modes
*    chs 13.9.95  errors corrected
*    chs 26.9.95  additional comments
*    chs 24.10.95 parameters M,V,e,f, mode 6
*    chs 5.1.96   round by rint(), -c and -f,
*                 using exp instead of pow with -e flag
*                 adjustment after exp !
*                 release 1.2
*
**************************************************************************
/


#include <string.h>
#include <stdio.h>
#include <math.h>

#include "ranlib.h"

/* prototypes: */

extern void fft_rif(double *data,int nn,int isign);

void  FGN_spectrum ( double *pow_spec, int size, double H);
double      FGN_B_est (double lambda, double H);
int   check_pow_of_2(long val);
double      tab_mean(double *tab, int n);
double      tab_var(double *tab, int n, double mean);

static char usgtxt[] = "\
######################################################################\n\
fft_fgn: Generation of Fractional Gaussian Noise by FFT-Algorithm\n\
Usage:\n\
      -h            this page\n\
      -H <value>    Hurst parameter, default 0.7\n\
      -M <value>    output mean value, default 0.0\n\
      -V <value>    output variance, default 1.0\n\
      -c            generate integer output by ceil(), default rint()\n\
      -f            generate integer output by floor(), default rint()\n\
      -e            apply final exponential transform\n\
      -o <filename> output data file (stdout is default)\n\
```

```
      -m <mode> set output:\n\
                  0: estimated power spectrum\n\
                  1: randomized power spectrum\n\
                  2: complex power spectrum\n\
                  3: complex power spectrum, re/im form\n\
                  4: output inverse fourier transform, re/im form\n\
                  5: real sample path (default)\n\
                  6: integer sample path\n\
      -n <value>    number of sample points (power of 2), default 4096\n\
      -s <value>    integer seed of random number generator\n\
      -v <level>    verbose level\n\
Warning: output mode 6 and parameter -e might destroy the\n\
         self-similar statistical properties of the output !\n\n\
       GMD FOKUS fft_fgn 1.2 96/01/05\n\
" ;
void usage() {
 fprintf(stderr, usgtxt);
 exit(0);
}

#define DEF_POINTS 4096
#define DEF_H 0.7
#define DEF_M 0.0
#define DEF_V 1.0
#define DEF_SEED1 31
#define SEED2 111 /* constant value ! */


int main(int argc,char *argv[])
{
      enum out_modes {SPECTRUM_EST = 0, SPECTRUM_FUZZED,
                              SPECTRUM_COMPLEX1, SPECTRUM_COMPLEX2,
                              SAMPLE_RE_IM, REAL_SAMPLE, INT_SAMPLE};

      char *oname = NULL;
      FILE *outfp = NULL;

      /* parameters: */
      int out_mode = REAL_SAMPLE;
      int n = DEF_POINTS;
      long seed1 = DEF_SEED1;
      long seed2 = SEED2;
      double H = DEF_H;
      double M = DEF_M;
      double V = DEF_V;
      int exp_flag = 0;
      int ceil_flag = 0;
      int floor_flag = 0;

      int n_2;
      int i;
      long int_val;

      double val,mean,var,re,im,factor,add_val;
      double *pow_spec,*z,*z_e;

      extern char *optarg;
      int   optind;

      int opt_err = 0;
      int verbose = 0;
```

```
      while ( (optind = getopt(argc, argv, "cefhm:n:r:s:o:v:H:M:V:") ) !=
EOF) {
            switch (optind) {
                  case 'e':
                        exp_flag = 1;
                        break;
                  case 'c':
                        ceil_flag = 1;
                        break;
                  case 'f':
                        floor_flag = 1;
                        break;
                  case 'h':
                        usage();
                        exit (0);
                        break;
                  case 'H':
                        if(sscanf(optarg,"%lf",&H) != 1) {
                              fprintf(stderr,"fft_fgn  parameter  H:  value
missing\n");
                              opt_err++;
                        }
                        break;
                  case 'M':
                        if(sscanf(optarg,"%lf",&M) != 1) {
                              fprintf(stderr,"fft_fgn  parameter  M:  value
missing\n");
                              opt_err++;
                        }
                        break;
                  case 'V':
                        if(sscanf(optarg,"%lf",&V) != 1) {
                              fprintf(stderr,"fft_fgn  parameter  V:  value
missing\n");
                              opt_err++;
                        }
                        break;
                  case 'm':
                        if(sscanf(optarg,"%d",&out_mode) != 1) {
                              fprintf(stderr,"fft_fgn  parameter  m:  value
missing\n");
                              opt_err++;
                        }
                        break;
                  case 'n':
                        if(sscanf(optarg,"%d",&n) != 1) {
                              fprintf(stderr,"fft_fgn  parameter  n:  value
missing\n");
                              opt_err++;
                        }
                        break;
                  case 'o':
                        oname = optarg;
                        break;
                  case 's':
                        if(sscanf(optarg,"%d",&seed1) != 1) {
                              fprintf(stderr,"fft_fgn  parameter  s:  seed
value missing\n");
                              opt_err++;
                        }
```

```
                        break;
                  case 'v':
                        if(sscanf(optarg,"%d",&verbose) != 1) {
                              fprintf(stderr,"fft_fgn  parameter  v:  value
missing\n");
                              opt_err++;
                        }
                        break;
                  default:
                        usage();
                        exit(1);

            }/*switch*/
      }/*while*/

      if (opt_err) {
            exit (1);
      }

      /* open output file: */
      if (oname == NULL)
            outfp = stdout;
      else {
            if ((outfp = fopen(oname,"w")) == NULL) {
                  fprintf(stderr,"fft_fgn: can't open outfp %s\n",oname);
                  exit (1);
            }
      }

      if( !check_pow_of_2(n) ) {
            fprintf(stderr,"fft_fgn error: number of sample points must be
power of 2 !\n");
            exit(1);
      }

      n_2 = n/2;

      /* allocate memory: */
      pow_spec = (double*) malloc( (n_2+1) * sizeof(double));
      z = (double*) malloc( (2*(n_2+1)) * sizeof(double));
      z_e = (double*) malloc(2 * (n+1) * sizeof(double));

      /* Approximate ideal power spectrum: */
      FGN_spectrum(pow_spec,n_2,H);

      if(out_mode == SPECTRUM_EST) {
            for(i = 1; i <= n_2; i++)
                  fprintf(outfp,"%g\n",pow_spec[i]);
            goto END;
      }

      /* Adjust for estimating power spectrum via periodogram: */
      setall(seed1,seed2);

      for(i = 1; i <= n_2; i++)
            pow_spec[i] *= genexp(1.0);

      if(out_mode == SPECTRUM_FUZZED) {
            for(i = 1; i <= n_2; i++)
                  fprintf(outfp,"%g\n",pow_spec[i]);
            goto END;
```

```
	}


	/* Construct corresponding complex numbers with random phase: */
	for(i = 1; i <= n_2; i++) {
		z[2*i] = sqrt( pow_spec[i] );
		z[2*i+1] = 2.0 * M_PI * ranf();
	}
	/* Last element should have zero phase: */
	z[2*n_2+1] = 0.0;

	/* Expand z to correspond to a Fourier transform of a real-valued
signal: */
	z_e[0] = 0.0;		/* consider mean value parameter ! */
	z_e[1] = 0.0;
	for(i = 1; i <= n_2; i++) {
		z_e[2*i] = z[2*i];
		z_e[2*i+1] = z[2*i+1];
	}
	for(i = n_2+1; i < n; i++) {
		z_e[2*i] = z[2*(n-i)];
		z_e[2*i+1] = - z[2*(n-i)+1];
	}

	if(out_mode == SPECTRUM_COMPLEX1) {
		for(i = 0; i < n; i++)
			fprintf(outfp,"%8.6g\t%8.6g\n",z_e[2*i],z_e[2*i+1]);
		goto END;
	}

	/* transform to RE/IM-Form: */
	for(i = 0; i < n; i++) {
		re = z_e[2*i] * cos(z_e[2*i+1]);
		im = z_e[2*i] * sin(z_e[2*i+1]);
		z_e[2*i] = re;
		z_e[2*i+1] = im;
	}

	if(out_mode == SPECTRUM_COMPLEX2) {
		for(i = 0; i < n; i++)
			fprintf(outfp,"%8.6g\t%8.6g\n",z_e[2*i],z_e[2*i+1]);
		goto END;
	}

	/* Inverse FFT gives sample path: */
	fft_rif(z_e,n,-1);

	if(out_mode == SAMPLE_RE_IM){
		for(i = 0; i < n; i++)
			fprintf(outfp,"%8.6g\t%8.6g\n",z_e[2*i],z_e[2*i+1]);
		goto END;
	}

	if(exp_flag) {
		/* apply exponential transform (to eliminate negative values
?): */

		/* mean should be about zero at this point !		*/
		/* variance is about the number of sample points !	*/
		/* exact adjustment follows after exp. transformation	*/
		factor = sqrt(1.0/n);
```

```
            for(i = 0; i < n; i++) z_e[2*i] = factor * z_e[2*i];
            if(verbose == 2) {
                  mean = tab_mean(z_e,n);
                  var = tab_var(z_e,n,mean);
                  fprintf(stderr,"before  exp  transformation:    mean  %f
variance %f\n",mean,var);
            }

            for(i = 0; i < n; i++) z_e[2*i] = exp(z_e[2*i]);
      }

      /* mean calculation: */
      mean = tab_mean(z_e,n);
      /* variance calculation: */
      var = tab_var(z_e,n,mean);
      if(verbose == 2) {
            fprintf(stderr,"before  variance  adjustment:  mean  %f  variance
%f\n",mean,var);
      }

      /* variance adjustment: */
      factor = sqrt(V/var);
      for(i = 0; i < n; i++) z_e[2*i] = factor * z_e[2*i];

      /* mean adjustment: */
      mean = tab_mean(z_e,n);
      if(verbose == 2) {
            var = tab_var(z_e,n,mean);
            fprintf(stderr,"before mean adjustment:       mean %f variance
%f\n",mean,var);
      }
      add_val = M - mean;
      for(i = 0; i < n; i++) z_e[2*i] = z_e[2*i] + add_val;

      if(out_mode == REAL_SAMPLE){
            for(i = 0; i < n; i++)
                  fprintf(outfp,"%8.6g\n",z_e[2*i]);
            goto END;
      }

      if(out_mode == INT_SAMPLE) {

            if(floor_flag)
                  for(i = 0; i < n; i++) z_e[2*i] = floor(z_e[2*i]);
            else if(ceil_flag)
                  for(i = 0; i < n; i++) z_e[2*i] = ceil(z_e[2*i]);
            else
                  for(i = 0; i < n; i++) z_e[2*i] = rint(z_e[2*i]);

            for(i = 0; i < n; i++)
                  fprintf(outfp,"%d\n",(int)z_e[2*i]);
      }




      END:

      if(verbose) {
            /* control mean and variance calculation:*/
            mean = tab_mean(z_e,n);
```

```
            var = tab_var(z_e,n,mean);

            fprintf(stderr,"fft_fgn: Fractional Gaussian Noise\n\
Hurst parameter:                 %f\n\
number of sample points:         %d\n\
seed of random number generator: %d\n",H,n,seed1);

            if(out_mode >= 5) {
                  fprintf(stderr,"\
mean parameter:                  %f\n\
variance parameter:              %f\n\
output mean value:               %f\n\
output variance:                 %f\n",M,V,mean,var);
            }

            fprintf(stderr,"\
output data:                     ");
            switch(out_mode) {
                  case SPECTRUM_EST:
                        fprintf(stderr,"estimated power spectrum\n");
                        break;
                  case SPECTRUM_FUZZED:
                        fprintf(stderr,"randomized power spectrum\n");
                        break;
                  case SPECTRUM_COMPLEX1:
                        fprintf(stderr,"complex power spectrum\n");
                        break;
                  case SPECTRUM_COMPLEX2:
                        fprintf(stderr,"complex   power   spectrum,   re-im
form\n");
                        break;
                  case SAMPLE_RE_IM:
                        fprintf(stderr,"Real Sample Path, re-im form\n");
                        break;
                  case REAL_SAMPLE:
                        fprintf(stderr,"Real Sample Path\n");
                        break;
                  case INT_SAMPLE:
                        fprintf(stderr,"Integer Sample Path\n");
                        if(floor_flag)     fprintf(stderr,"truncated     by
floor()\n");
                        else   if(ceil_flag)   fprintf(stderr,"truncated   by
ceil()\n");
                        else fprintf(stderr,"truncated by rint()\n");
                        break;
                  default: break;
            }/*switch*/
            if(exp_flag) fprintf(stderr,"exponential transformation applied
!\n");
      }/*verbose*/

      free ( pow_spec );
      free ( z );
      free ( z_e );
      exit(0);

}/*main*/

/*********************************************************************
***/
void FGN_spectrum ( double *pow_spec, int n, double H)
```

```
{
/*
 Returns an approximation of the power
 spectrum for fractional Gaussian noise
 at the given frequencies lambda and
 the given Hurst parameter H.
*/
      int i;
      double lambda,fact1,a,b,c,g;

      /* the result of lgamma will always be positive: */
      g = lgamma(2.0 * H + 1.0);

      fact1 = 2.0 * sin(M_PI * H) * exp(g);

      for(i = 1; i < n+1; i++) {
            lambda = (M_PI * i) / n;
            a = fact1 * (1.0 - cos(lambda));
            b = pow(lambda,(-2.0 * H - 1.0));
            c = FGN_B_est(lambda, H);
            pow_spec[i] = a * (b + c);
      }

}/*FGN_spectrum*/


/*************************************************************************
***/
double FGN_B_est (double lambda, double H)
{
/*
 Returns the estimate for
 B(lambda,H).
*/
      int k;
      double d, dprime, sum1,sum2,result;
      double a[5],b[5]; /* index 0 never used ! */

      d = -2.0 * H - 1.0;
      dprime = -2.0 * H;

      for(k=1;k<5;k++) {
            a[k] = 2.0 * k * M_PI + lambda;
            b[k] = 2.0 * k * M_PI - lambda;
      }

      sum1 = 0.0;
      for(k=1;k<4;k++) {
            sum1 += pow(a[k],d);
            sum1 += pow(b[k],d);
      }
      sum2 = 0.0;
      for(k=3;k<5;k++) {
            sum2 += pow(a[k],dprime);
            sum2 += pow(b[k],dprime);
      }
      result = sum1 + (sum2 / (8.0 * M_PI * H));
      return(result);

}/*FGN_B_est*/
```

```
/*************************************************************************
***/
int check_pow_of_2(long val)
{int i;
      /* check for nn = power of 2: */
      for(i=2;i<=32;i++) {
            if(pow(2.0,i) == val) return(1);
      }
      return(0);
}/*check_pow_of_2*/

/*************************************************************************
***/
double tab_mean(double *tab, int n)
{
      int i;
      double mean;

      mean = 0.0;
      for(i = 0; i < n; i++) {
            mean += tab[2*i];
      }
      mean /= n;
      return(mean);
}

/*************************************************************************
***/
double tab_var(double *tab, int n, double mean)
{
      int i;
      double var;

      var = 0.0;
      for(i = 0; i < n; i++) {
            var += ((tab[2*i] - mean) * (tab[2*i] - mean));
      }
      var /= n;
      return(var);
}
```

ANEXO E – Exemplo de Código para Gerador de Tráfego FTP

Este código foi extraído do simulador *ns*, disponível em [NS, 2003], arquivo `ns-source.tcl`, escrito em linguagem OTcl (*Object Tool Command Language*). O código implementa a geração FTP como um fluxo contínuo de dados (um *stream*), ativando o objeto TCL para que envie segmentos continuamente (através do método para o objeto TCL `'send -1').`

```
#
# Copyright (c) 1996 Regents of the University of California.
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
# 3. All advertising materials mentioning features or use of this software
#    must display the following acknowledgement:
#     This product includes software developed by the MASH Research
#     Group at the University of California Berkeley.
# 4. Neither the name of the University nor of the Research Group may be
#    used to endorse or promote products derived from this software without
#    specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
#  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
#   FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
#  HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.
#
#  @(#)  $Header:  /nfs/jade/vint/CVSROOT/ns-2/tcl/lib/ns-source.tcl,v  1.25
2000/11/01 21:47:59 xuanc Exp $
#

#  NOTE:  Could consider renaming this file to ns-app.tcl and moving the
#  backward compatible code to ns-compat.tcl
```

```
#set ns_telnet(interval) 1000ms
#set ns_bursty(interval) 0
#set ns_bursty(burst-size) 2
#set ns_message(packet-size) 40

Class Application/FTP -superclass Application

Application/FTP instproc init {} {
      $self next
}

Application/FTP instproc start {} {
      [$self agent] send -1
}

# Causes TCP to send no more new data.
Application/FTP instproc stop {} {
      [$self agent] advance 0
      [$self agent] close
}

Application/FTP instproc send {nbytes} {
      [$self agent] send $nbytes
}

# For sending packets.  Sends $pktcnt packets.
Application/FTP instproc produce { pktcnt } {
      [$self agent] advance $pktcnt
}

# For sending packets.  Sends $pktcnt more packets.
Application/FTP instproc producemore { pktcnt } {
      [$self agent] advanceby $pktcnt
}

# Backward compatibility
Application/Traffic instproc set args {
      $self instvar packetSize_ rate_
      if { [lindex $args 0] == "packet_size_" } {
            if { [llength $args] == 2 } {
                  set packetSize_ [lindex $args 1]
                  return
            } elseif { [llength $args] == 1 } {
                  return $packetSize_
            }
      }
      eval $self next $args
}
# Helper function to convert rate_ into an interval_
#Hicked to generate pulse UDP traffic (seperate rate_ and interval_).
Application/Traffic/CBR instproc set args {
      $self instvar packetSize_ rate_
      if { [lindex $args 0] == "interval_" } {
            if { [llength $args] == 2 } {
                  set ns_ [Simulator instance]
                  set interval_ [$ns_ delay_parse [lindex $args 1]]
                  $self set rate_ [expr $packetSize_ * 8.0/$interval_]
                  return
            } elseif { [llength $args] == 1 } {
                  return [expr $packetSize_ * 8.0/$rate_]
            }
```

```
	}
	eval $self next $args
}

#
# Below are backward compatible components
#

Class Agent/CBR -superclass Agent/UDP
Class Agent/CBR/UDP -superclass Agent/UDP
Class Agent/CBR/RTP -superclass Agent/RTP
Class Agent/CBR/UDP/SA -superclass Agent/SA

Agent/SA instproc attach-traffic tg {
	$tg attach-agent $self
	eval $self cmd attach-traffic $tg
}

Agent/CBR/UDP instproc attach-traffic tg {
	$self instvar trafgen_
	$tg attach-agent $self
	set trafgen_ $tg
}

Agent/CBR/UDP instproc done {} { }

Agent/CBR/UDP instproc start {} {
	$self instvar trafgen_
	$trafgen_ start
}

Agent/CBR/UDP instproc stop {} {
	$self instvar trafgen_
	$trafgen_ stop
}

Agent/CBR/UDP instproc advance args {
	$self instvar trafgen_
	eval $trafgen_ advance $args
}

Agent/CBR/UDP instproc advanceby args {
	$self instvar trafgen_
	eval $trafgen_ advanceby $args
}

Agent/CBR instproc init {} {
	$self next
	$self instvar trafgen_ interval_ random_ packetSize_ maxpkts_
	# The following used to be in ns-default.tcl
	set packetSize_ 210
	set random_ 0
	set maxpkts_ 268435456
	set interval_ 0.00375
	set trafgen_ [new Application/Traffic/CBR]
	$trafgen_ attach-agent $self
	# Convert packetSize_ and interval_ to trafgen_ rate_
	$trafgen_ set rate_ [expr $packetSize_ * 8.0/ $interval_]
	$trafgen_ set random_ [$self set random_]
	$trafgen_ set maxpkts_ [$self set maxpkts_]
	$trafgen_ set packetSize_ [$self set packetSize_]
```

```
      # The line below is needed for backward compat with v1 test scripts
        if {[Simulator set nsv1flag] == 0} {

          puts     "using     backward     compatible     Agent/CBR;     use
Application/Traffic/CBR instead"
      }
}



Agent/CBR instproc done {} { }

Agent/CBR instproc start {} {
      $self instvar trafgen_
      $trafgen_ start
}

Agent/CBR instproc stop {} {
      $self instvar trafgen_
      $trafgen_ stop
}

Agent/CBR instproc advance args {
      $self instvar trafgen_
      eval $trafgen_ advance $args
}

Agent/CBR instproc advanceby args {
      $self instvar trafgen_
      eval $trafgen_ advanceby $args
}

# Catches parameter settings for overlying traffic generator object
Agent/CBR instproc set args {
      $self instvar interval_ random_ packetSize_ maxpkts_ trafgen_
      if { [info exists trafgen_] } {
            if { [lindex $args 0] == "packetSize_" } {
                  if { [llength $args] == 2 } {
                        $trafgen_ set packetSize_ [lindex $args 1]
                        set packetSize_ [lindex $args 1]
                        # Recompute rate
                        $trafgen_  set  rate_  [expr  $packetSize_  *  8.0/
$interval_]
                              return
                  } elseif { [llength $args] == 1 } {
                        return $packetSize_
                  }
            } elseif { [lindex $args 0] == "random_" } {
                  if { [llength $args] == 2 } {
                        $trafgen_ set random_ [lindex $args 1]
                        set random_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                        return $random_
                  }
            } elseif { [lindex $args 0] == "maxpkts_" } {
                  if { [llength $args] == 2 } {
                        $trafgen_ set maxpkts_ [lindex $args 1]
                        set maxpkts_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
```

```
                        return $maxpkts_
                  }
            } elseif { [lindex $args 0] == "interval_" } {
                  if { [llength $args] == 2 } {
                        set ns_ [Simulator instance]
                        set interval_ [$ns_ delay_parse [lindex $args 1]]
                        # Convert interval_ to rate for trafgen_
                        $trafgen_ set rate_ [expr $packetSize_ * 8.0/
$interval_]
                        return
                  } elseif { [llength $args] == 1 } {
                        return $interval_
                  }
            }
      }
      eval $self next $args
}

Class Traffic/Expoo -superclass Application/Traffic/Exponential
Class Traffic/Pareto -superclass Application/Traffic/Pareto
Class Traffic/RealAudio -superclass Application/Traffic/RealAudio
Class Traffic/Trace -superclass Application/Traffic/Trace

# These instprocs are needed to map old Traffic/* type variables
Traffic/Expoo instproc set args {
      $self instvar packetSize_ burst_time_ idle_time_ rate_
      if { [lindex $args 0] == "packet-size" } {
            if { [llength $args] == 2 } {
                  $self set packetSize_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $packetSize_
                  }
      } elseif { [lindex $args 0] == "burst-time" } {
            if { [llength $args] == 2 } {
                  $self set burst_time_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $burst_time_
                  }
      } elseif { [lindex $args 0] == "idle-time" } {
            if { [llength $args] == 2 } {
                  $self set idle_time_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $idle_time_
                  }
      } elseif { [lindex $args 0] == "rate" } {
            if { [llength $args] == 2 } {
                  $self set rate_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $rate_
                  }
      }
      eval $self next $args
}

Traffic/Pareto instproc set args {
      $self instvar packetSize_ burst_time_ idle_time_ rate_ shape_
      if { [lindex $args 0] == "packet-size" } {
```

```
            if { [llength $args] == 2 } {
                  $self set packetSize_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $packetSize_
                  }
      } elseif { [lindex $args 0] == "burst-time" } {
            if { [llength $args] == 2 } {
                  $self set burst_time_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $burst_time_
                  }
      } elseif { [lindex $args 0] == "idle-time" } {
            if { [llength $args] == 2 } {
                  $self set idle_time_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $idle_time_
                  }
      } elseif { [lindex $args 0] == "rate" } {
            if { [llength $args] == 2 } {
                  $self set rate_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $rate_
                  }
      } elseif { [lindex $args 0] == "shape" } {
            if { [llength $args] == 2 } {
                  $self set shape_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $shape_
                  }
      }
      eval $self next $args
}

Traffic/RealAudio instproc set args {
      $self instvar packetSize_ burst_time_ idle_time_ rate_
      if { [lindex $args 0] == "packet-size" } {
            if { [llength $args] == 2 } {
                  $self set packetSize_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $packetSize_
                  }
      } elseif { [lindex $args 0] == "burst-time" } {
            if { [llength $args] == 2 } {
                  $self set burst_time_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $burst_time_
                  }
      } elseif { [lindex $args 0] == "idle-time" } {
            if { [llength $args] == 2 } {
                  $self set idle_time_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $idle_time_
                  }
```

```
      } elseif { [lindex $args 0] == "rate" } {
            if { [llength $args] == 2 } {
                  $self set rate_ [lindex $args 1]
                        return
                  } elseif { [llength $args] == 1 } {
                  return $rate_
                  }
      }
      eval $self next $args
}

Class Source/FTP -superclass Application
Source/FTP set maxpkts_ 268435456

Source/FTP instproc attach o {
      $self instvar agent_
      set agent_ $o
      $self attach-agent $o
}

Source/FTP instproc init {} {
      $self next
      $self instvar maxpkts_ agent_
      set maxpkts_ 268435456
}

Source/FTP instproc start {} {
      $self instvar agent_ maxpkts_
      $agent_ advance $maxpkts_
}

Source/FTP instproc stop {} {
      $self instvar agent_
      $agent_ advance 0
}

Source/FTP instproc produce { pktcnt } {
      $self instvar agent_
      $agent_ advance $pktcnt
}

Source/FTP instproc producemore { pktcnt } {
      $self instvar agent_
      $agent_ advanceby $pktcnt
}


#
# For consistency with other applications
#
Class Source/Telnet -superclass Application/Telnet

Source/Telnet set maxpkts_ 268435456

Source/Telnet instproc attach o {
      $self instvar agent_
      set agent_ $o
      $self attach-agent $o
}
```

ÍNDICE

A

B

C

D

E

F

G

H

I

J

L

M

O

P

Q

R

S

T

V

W

X