UNIVERSIDADE SALVADOR – UNIFACS
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO
MESTRADO PROFISSIONAL EM SISTEMAS E COMPUTAÇÃO

MARCOS PORTNOI

UM PROTÓTIPO DE SIMULADOR DE REDES DE COMPUTADORES PARA APLICAÇÕES ESPECÍFICAS BASEADAS NO PROTOCOLO MPLS

Salvador – BA
2007

MARCOS PORTNOI

UM PROTÓTIPO DE SIMULADOR DE REDES DE COMPUTADORES PARA APLICAÇÕES ESPECÍFICAS BASEADAS NO PROTOCOLO MPLS

Dissertação apresentada ao Programa de Pós-Graduação em Sistemas e Computação, Universidade Salvador – UNIFACS, como requisito parcial para obtenção do grau de Mestre em Sistemas e Computação.

Orientador: Prof. Dr. Joberto Sergio Barbosa Martins

Salvador – BA

2007

FICHA CATALOGRÁFICA
(Elaborada pelo Sistema de Bibliotecas da Universidade Salvador - UNIFACS)

Portnoi, Marcos
Um protótipo de simulador de redes de computadores para aplicações específicas baseadas no protocolo MPLS /Marcos Portnoi. - 2007.
118 f.

Dissertação (Mestrado) - Universidade Salvador – UNIFACS. Mestrado em Sistemas e Computação, 2007.
Orientador: Prof. Dr. Joberto Sergio Barbosa Martins

1. Redes de computadores. 2. Simulador de rede. 3. Recuperação de falhas. 4. Avaliação de desempenho. 5. Engenharia de tráfego. I. Martins, Joberto Sergio Barbosa, orient. II. Título.

CDD: 004.6

PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO
MESTRADO EM SISTEMAS E COMPUTAÇÃO

MARCOS PORTNOI

UM PROTÓTIPO DE SIMULADOR DE REDES DE COMPUTADORES PARA APLICAÇÕES ESPECÍFICAS BASEADAS NO PROTOCOLO MPLS

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Redes de Computadores, Universidade Salvador - UNIFACS, do Curso de Mestrado Profissional em Redes de Computadores, pela seguinte banca examinadora:

Prof. Joberto Sérgio Barbosa Martins (Orientador)
Doutor em Informatique, Universite de Paris VI (Pierre et Marie Curie), U.P. VI, França.
Universidade Salvador - UNIFACS

Profa. Maria Izabel Cavalcanti Cabral
Doutora em Engenharia Elétrica, Universidade Federal da Paraíba (UFPB)
Universidade Federal de Campina Grande - UFCG

Prof. José Augusto Suruagy Monteiro
Doutor em Computer Scince, University of Califórnia Los Angeles, U.C.L..A, Estados Unidos.
Universidade Salvador - UNIFACS

Salvador, 04 de outubro de 2007.

AGRADECIMENTOS


```
void dankeschoen() {
        printf("D. MC. P. JM. SB. usw.");
        printf("\n");
        dankeschoen();
}
```

“Na main moosa na pharoah
Na main jagan na vich saun
Na main aatish na main paun
Na main rahnda vich Nadaun
Na main baitthan na vich bhaun
Bulle shah kharha hai kaun”

-- Rabbi Shergill

RESUMO

Várias aplicações de redes de computadores demandam requisitos de Qualidade de Serviço (*Quality of Service* – QoS). Aplicações de Voz sobre IP (VoIP), por exemplo, usualmente têm certa tolerância definida para atrasos e suportam baixas perdas de pacotes. Os procotolos convencionais de roteamento e encaminhamento usados nas redes, como o IP e OSPF, trabalham na filosofia de melhor esforço. Ou seja, métricas como capacidade de enlaces, características de tráfego e ainda reserva de recursos são conceitos não considerados. Apesar de serem estes protocolos robustos no tocante a recuperação de falhas na rede, como por exemplo falhas de enlace, o mecanismo de recuperação de falhas não leva em consideração o tempo de recuperação, que impacta diretamente sobre o desempenho das aplicações. O protocolo MPLS pode prover mecanismos que, em conjunto com o protocolo de sinalização RSVP-TE, possibilitam ser usados como ferramenta de Engenharia de Tráfego numa rede de computadores. Esta Engenharia de Tráfego pode compreender controle de fluxos de tráfego dentro da rede, controle de utilização de recursos, roteamento por restrição e ainda oferece uma mecânica de recuperação rápida em caso de falha. Estes fatores podem ser usados de modo a garantir níveis de QoS para diferentes aplicações. Partindo destes conceitos e de um primeiro interesse em examinar o efeito da recuperação rápida sobre o desempenho de uma aplicação usando simulação, apreciam-se alguns simuladores disponíveis. Decide-se, então, pela construção de um protótipo de simulador de redes de computadores, denominado TARVOS *Computer Networks Simulator*, orientado para a simulação da funcionalidade de recuperação rápida em redes MPLS com RSVP-TE. Os detalhes construtivos do TARVOS são demonstrados e seu funcionamento exposto, seguindo com um roteiro de modelagem de simulações. Além disso, como investigação e validação do protótipo confeccionado, estuda-se o efeito de falhas de enlace no desempenho de uma aplicação VoIP.

Palavras-chave: redes de computadores, simulador, recuperação de falhas, avaliação de desempenho, engenharia de tráfego, qualidade de serviço, QoS, MPLS, RSVP-TE, VoIP, recuperação rápida, simulação.

ABSTRACT

Several applications for computer networks require certain levels of Quality of Service (QoS). Voice over IP (VoIP) applications, for instance, generally tolerate a certain amount of delay and support a low packet loss. Conventional routing and forwarding protocols, used in computer networks, such as IP and OSPF, work based on the best effort philosophy. I.e., metrics like link capacity, traffic characteristics, and moreover resource reservation are concepts not taken into consideration. Although these protocols are robust when it comes to network fault recovery, for instance link failures, the mechanism responsible for fault recovery does not take into account the recovery time, which impacts directly onto the performance of applications. The MPLS protocol, together with the signaling protocol RSVP-TE, provides functionalities that can be used as tools for Traffic Engineering in a computer network, comprising traffic flow control, resource utilization control, constraint routing, and also implementing an algorithm for fast recovery in case of failure. All these factors can be used as to guarantee QoS levels for different applications. From these concepts and from the initial interest in examining the effect of rapid recovery on an application's performance, by using simulation, a number of available simulators are evaluated. This leads to the building of a prototype for a computer networks simulator, named TARVOS Computer Networks Simulator, oriented to simulate the fast recovery functionality in MPLS and RSVP-TE networks. TARVOS' constructive details are demonstrated and its functionalities explained, followed by a guide to model simulations. Additionaly, in order to investigate and validate the prototype, the study of the effect of link failures on the performance of a VoIP application is presented.

Keywords: computer networks, simulator, fault recovery, performance analysis, traffic engineering, quality of service, QoS, MPLS, RSVP-TE, VoIP, fast recovery, rapid recovery , simulation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BGP	*Border Gateway Protocol*
CBR	*Constant Bit Rate*
CBS	*Committed Bucket Size*
CIR	*Committed Information Rate*
CSPF	*Constraint Shortest Path First*
FCFS	*First Come, First Served*
FIFO	*First In, First Out*
HDTV	*High Definition Television*
IEEE	*Institute of Electrical and Electronic Engineers*
IP	*Internet Protocol*
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*
LER	*Label Edge Router*
LIB	*Label Information Base*
LRD	*Long Range Dependence*
LSP	*Label Switched Path*
LSR	*Label Switched Router*
MP	*Merge Point*
MPEG	*Moving Picture Experts Group*
MPLS	*MultiProtocol Label Switching*
OSPF	*Open Shortest Path First*
OSPF-TE	*Open Shortest Path First – Traffic Engineering*
PCM	*Pulse Code Modulation*
PDU	*Protocol Data Unit*
PIR	*Peak Information Rate*
PQ	*Priority Queue*

QoS	*Quality of Service*
RIP	*Routing Information Protocol*
RR	*Round Robin*
RSVP-TE	*ReSource reserVation Protocol – Traffic Engineering*
RTCP	*Real Time Control Protocol*
RTP	*Real Time Transport Protocol*
SRD	*Short Range Dependence*
TCP	*Transmission Control Protocol*
TTL	*Time To Live*
UDP	*User Datagram Protocol*
VoIP	*Voice over Internet Protocol*
WAN	*Wide Area Network*
WFQ	*Weighted Fair Queueing*
WRR	*Weighted Round Robin*
WWW	*World Wide Web*

SUMÁRIO

1. Introdução

1.1. Panorama

Serviços baseados em tecnologia Web (*Web Services*) têm experimentado crescimento considerável. Na internet, diversos sítios provêm aplicações para o usuário final, a exemplo de planilhas, editores de texto, agendas e gerenciadores de tarefas. Aplicações de telefonia (VoIP ou *Voice over IP*) e vídeo sobre redes de computadores também tendem a se popularizar, graças à introdução de serviços de baixo custo, a exemplo do Skype (SKYPE, 2006), e sítios na *web* baseados em conteúdo criado pelo usuário, para os quais o YouTube (YOUTUBE, 2006) é um forte representante.

Vários destes serviços demandam certas restrições ou requisitos mínimos de Qualidade de Serviço (*Quality of Service* – QoS). Aplicações de VoIP usualmente têm uma certa tolerância definida para atrasos e suportam apenas baixas perdas de pacotes. *Video Streaming,* ou fluxo de vídeo enviado continuamente via rede de computador, pode admitir alguma perda

de pacotes sem decréscimo significativo da qualidade, porém é bastante sensível a atrasos e variação do atraso (*jitter*).

Uma das causas de perda de pacotes é uma falha num enlace de rede. Um enlace rompido ou não funcional resultará no descarte de pacotes que deveriam trafegar por ele até que uma nova rota seja construída e disponibilizada para a rede. O processo de detecção da falha no enlace, construção ou escolha de uma nova rota que evite tal enlace e distribuição desta informação para os pontos necessários da rede é denominado, genericamente, *Recuperação de Falhas* (YI; CHUNG-HORNG; SRINIVASAN, 2004). Todo o procedimento de recuperação da falha causará um atraso extra para as aplicações que enviam dados pelo enlace defeituoso. O protocolo IP (*Internet Protocol*), usado na internet, é robusto e capaz de restabelecer a conectividade quando submetido a vários tipos de mau funcionamento de elementos da rede (falhas em enlaces, falhas nos roteadores, *switches*, placas de rede, por exemplo); no entanto, é um protocolo do tipo "melhor esforço" (*best effort*). Isto significa que o IP não garante ou provê qualquer forma de QoS. Assim, o tempo em que o IP leva para restabelecer a conectividade entre dois pontos pode ultrapassar os limites requeridos por uma aplicação do usuário (PETERSSON, 2005).

Mais ainda, os protocolos convencionais de roteamento usados em conjunto com o IP, como o BGP (*Border Gateway Protocol*) (REKHTER; LI; HARES, 2006) e OSPF (*Open Shortest Path First*) (MOY, 1998), de acordo com a filosofia do melhor esforço, não levam em consideração, quando calculando e decidindo sobre rotas, métricas como capacidade do enlace e

características de tráfego. Isso possivelmente resulta em caminhos sub- ou sobreutilizados, revertendo em desperdício de recursos no primeiro caso, e congestionamento, perda de pacotes e atrasos, no segundo caso. Destarte, mecanismos precisam ser adicionados ao TCP/IP (*Transmission Control Protocol/Internet Protocol*) convencional de maneira a obter controle sobre os fluxos de tráfego dentro da rede e assim otimizar o desempenho e utilização de recursos; este é um dos objetivos da Engenharia de Tráfego (*Traffic Engineering* ou TE) (AWDUCHE *et al*, 1999).

O protocolo MPLS (*MultiProtocol Label Switching*) (ROSEN; VISWANATHAN; CALLON, 2001) provê um mecanismo que, conjuntamente com um protocolo de sinalização como o RSVP-TE (*ReSource reserVation Protocol – Traffic Engineering*) (AWDUCHE *et al*, 2001), pode ser usado como ferramenta para implementar a Engenharia de Tráfego numa rede de computadores. As potencialidades do MPLS e RSVP-TE compreendem o roteamento por restrição (*constraint routing*) – que será definido adiante – tunelamento e mecanismos para re-roteamento e recuperação rápida, além das funcionalidades básicas destes dois protocolos. Estas potencialidades podem ser dispostas de modo a garantir níveis de QoS para diferentes aplicações.

1.2. Objetivos da Dissertação

O empenho inicial em analisar tais protocolos, em especial o engenho de recuperação rápida de falhas em redes MPLS com RSVP-TE e seu impacto no desempenho de uma aplicação (sob uma falha na rede), foi

seguido de uma averiguação de ferramentas de simulação existentes. Isso se fez necessário pois o intuito era avaliar o impacto no desempenho da aplicação com uso de um simulador, e tal simulador precisava satisfazer uma série de requisitos desejados (e que serão ponderados em capítulo adiante neste texto). A averiguação dos simuladores culminou, contudo, em uma série de incentivos a se desenvolver uma nova ferramenta de simulação.

A presente Dissertação exibe, assim, partindo do objeto inicial de examinar as funcionalidades dos protocolos MPLS e RSVP-TE, a construção de um protótipo de simulador de redes de computadores, orientado para a simulação do mecanismo de recuperação rápida em redes MPLS com RSVP-TE. Além disso, estuda, como investigação e validação da prototipação, o efeito de falhas de enlace no desempenho de aplicações VoIP. A metodologia adotada para a investigação é, portanto, o uso do protótipo de simulador de redes de computadores de modo a obter as medidas de desempenho a partir de uma topologia de teste. Os parárafos na seção seguinte descrevem os capítulos seguintes a esta introdução, que compõem este documento.

1.3. Estrutura desta Dissertação

O **Capítulo 2** aborda as características e funcionamento gerais do protocolo MPLS.

O **Capítulo 3** trata dos conceitos pertinentes ao Roteamento Baseado em Restrições, funcionalidade oferecida pelo protocolo RSVP-TE.

O **Capítulo 4** lista uma série de simuladores de redes de computadores que foram pesquisados ou testados e suas peculiaridades, que serviram como motivador para a construção do protótipo aqui descrito.

No **Capítulo 5**, descreve-se a construção e atributos do protótipo, as funções incorporadas e o comportamento geral da ferramenta.

O **Capítulo 6** expõe os passos para se confeccionar uma simulação com a ferramenta prototipada, demonstrando vários dos recursos programados.

O **Capítulo 7** traz um estudo, feito com o simulador, com vistas a investigar o efeito de falhas de enlace no desempenho de uma aplicação VoIP e, concomitantemente, validar o funcionamento do protótipo.

O **Capítulo 8**, finalmente, conclui o documento e indica trabalhos futuros.

Os termos em inglês, em regra, não serão traduzidos neste documento.

2. MPLS: Visão Geral

Um dos objetivos do protocolo MPLS (*MultiProtocol Label Switching*) (ROSEN; VISWANATHAN; CALLON, 2001) é otimizar o desempenho dos roteadores envolvidos em comutação dos pacotes IP (objetivo importante quando da idealização do protocolo, mas minorado frente ao ganho de processamento dos roteadores). Esta melhora é conseguida evitando a consulta, por parte dos roteadores, nas extensas tabelas de rotas para cada pacote IP comutado. O MPLS encaminha os pacotes baseado em rótulos (*labels*) que são anexados ao cabeçalho destes mesmos pacotes. Estruturalmente, o MPLS situa-se entre as camadas de enlace e a camada de rede (Figura 1). Um domínio de rede funcionando sob MPLS pode encaminhar os pacotes IP de maneira transparente e também outros pacotes já etiquetados com rótulos MPLS oriundos de outros domínios de rede MPLS, característica esta chamada de *empilhamento de rótulos* (ROSEN; VISWANATHAN; CALLON, 2001; NORTEL, 2005).

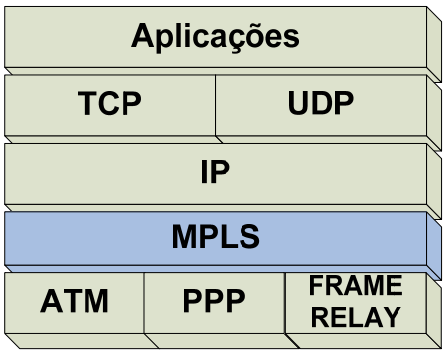

Figura 1: Pilha de protocolos de rede com MPLS.

O ganho de desempenho no encaminhamento (*forwarding*) é especialmente sensível em roteadores com baixa capacidade de processamento. Entretanto, roteadores de alto poder de processamento não são minoria e, devido à evolução tecnológica do hardware, tendem a se tornar ubíquos. Estes roteadores, que chegam a rotear na ordem de milhões de pacotes por segundo, apresentam bom desempenho no encaminhamento dos pacotes, mesmo consultando tabelas comuns de endereços IP. Assim, o foco de interesse do MPLS desloca-se do encaminhamento puro e simples para a possibilidade de, através deste protocolo, implementar também Engenharia de Tráfego para a rede.

2.1. Como Opera o MPLS

A operação básica do MPLS pode ser resumida conforme segue:

- As rotas dentro do domínio MPLS são previamente estabelecidas através de um protocolo de roteamento, como o OSPF (*Open Shortest Path First*) ou BGP (*Border Gateway Protocol*). O MPLS

utilizará estas rotas para criar as LSPs[1] (*Label Switched Paths*). As LSPs correspondem aos caminhos seguidos pelos pacotes.

- Os roteadores da borda do domínio MPLS recebem o nome de LER – *Label Edge Routers*. Os roteadores de núcleo do domínio são denominados LSR – *Label Switched Routers*.

- Um novo pacote IP adentra o domínio MPLS através de um LER. O domínio confere a este pacote uma classe FEC (*Forward Equivalent Class*). Grupos de pacotes pertencem a uma determinada FEC de acordo com algum parâmetro, como mesmo destino, mesma classe QoS, mesma interface de entrada, etc.

- Um algoritmo de distribuição de rótulos é então executado, de forma a estabelecer uma LSP associada à FEC. Cada roteador, começando pelo LER, requisita do próximo um rótulo a fim de marcar o pacote a ser encaminhado. Ao final da execução do algoritmo de distribuição de rótulos, cada roteador terá construído uma tabela associando FECs a rótulos e interfaces de entrada e saída. Esses algoritmos de distribuição de rótulos podem ser, por exemplo, o LDP (*Label Distribution Protocol*) (ANDERSSON *et al*, 2001) – o mais simples, o CR-LDP (JAMOUSSI *et al*, 2002; PORTNOI, 2005), o RSVP-TE.

[1] Neste texto, a sigla LSP é convencionada como do gênero feminino, portanto pedindo o artigo "a". A convenção baseou-se meramente na sonoridade da sigla.

- Um LSR, ao receber um pacote, lerá o rótulo afixado a este pacote e consultará a tabela de rótulos MPLS. Obterá a novo rótulo para marcar o pacote, e assim o roteador retirará o rótulo recebido a fim de substituí-lo pelo novo rótulo. O próximo LSR da rota LSP receberá o pacote, identificará o caminho através do rótulo a ele afixado, substituirá pelo rótulo constante em sua tabela e assim por diante, até que o último LER seja atingido. O rótulo MPLS é então retirado e o pacote é encaminhado convencionalmente (MARTINS, 2004a).

Em sua concepção tradicional, as redes baseadas em IP funcionam sob o mecanismo de "melhor esforço" (*best effort* ou BE). Neste modo, os pacotes são encaminhados para seu destino conforme for possível pela rede, sem quaisquer garantias de atraso, largura de banda, variação do atraso (*jitter*) ou mesmo que efetivamente irão atingir o destino. O MPLS, em combinação com um protocolo de sinalização e distribuição de rótulos (como o RSVP-TE), oferece como ferramenta de engenharia de tráfego a capacidade de estabelecer rotas onde restrições ou requisitos de recursos são levados em conta (ABOUL_MAGT; JAMOUSSI, 2001).

2.2. Resumo do Capítulo

Este breve Capítulo traçou um sumário das características do protocolo MPLS. Ele situa-se entre as camadas de rede e enlace na pilha de protocolos, e objetiva substituir o encaminhamento de pacotes baseados em extensas tabelas de rotas por um mais simplificado, baseado em rótulos: os

rótulos funcionam como índices nas tabelas de encaminhamento. Os protocolos usuais de encaminhamento, como o IP, podem continuar sendo usados, mas são dispensados dentro do domínio MPLS, onde somente opera o encaminhamento por comutação de rótulos. Prossegue-se com uma descrição da comutação por rótulos e conclui-se com a utilidade do MPLS, em conjunto com um protocolo de sinalização, como ferramenta de engenharia de tráfego.

3. Roteamento Baseado em Restrições ou Requisitos

O estabelecimento de rotas dentro de um domínio, através de protocolos como o OSPF e RIP (*Routing Information Protocol*) (MALKIN, 1998), é feito baseado num algoritmo que otimiza ou minimiza uma métrica particular. No RIP, por exemplo, a métrica é o número de saltos (*hops*) ou nós. O RIP usa o algoritmo Bellman-Ford para computar, dentre vários caminhos, aquele que possui o menor número de nós ou *hops*. Para o OSPF, a métrica é estabelecida administrativamente, ou seja, o administrador ou gerente da rede estabelece, para cada enlace, a métrica de desejo. O OSPF, então, recorre ao algoritmo Dijkstra a fim de calcular uma rota que minimize a métrica definida, sendo que a métrica total é a soma das métricas individuais de cada enlace, definidas pelo administrador da rede (DAVIE; REKHTER, 2000; PORTNOI, 2005).

O roteamento baseado em restrições pode ser assim formalmente definido (DAVIE; REKHTER, 2000). Considere-se uma rede representada por um grafo (V, E), onde V é o conjunto de nós e E é o conjunto de enlaces interconectando estes nós. Associado a cada enlace existe um número de

atributos. Para cada par de nós no grafo, há um conjunto de restrições que têm de ser satisfeitas pela rota desde o primeiro nó do par até o segundo. Este conjunto de restrições é expresso em termos de atributos dos enlaces, normalmente conhecido apenas pelo primeiro nó do par. O objetivo do roteamento baseado em restrições é computar um caminho ou rota de um certo nó a outro, de modo que a rota não viole as restrições e seja ótima em relação a alguma métrica. Assim que a rota é computada, o roteamento baseado em restrições é responsável por estabelecer e manter o encaminhamento através da rota.

Os algoritmos convencionais de roteamento IP almejam encontrar rotas que minimizem ou otimizem uma certa métrica escalar (como, por exemplo, o número de saltos). O roteamento baseado em restrições busca encontrar uma rota que otimize uma métrica e, ao mesmo tempo, não viole um conjunto de restrições. É esta característica de encontrar uma rota que obedeça a um conjunto de restrições que distingue o roteamento convencional IP do roteamento baseado em restrições.

Um tipo de restrição pode ser a manutenção de certas características de desempenho, como, por exemplo, encontrar uma rota que mantenha determinada largura de banda mínima disponível. A rota computada deve, pois, possuir a quantidade de banda mínima em todos os enlaces ao longo do caminho. Observar que cada rota estabelecida dentro de uma rede pode ter restrições diferentes de largura de banda (aliás, pode ter um conjunto de restrições completamente diferente de outras rotas na mesma rede).

Uma restrição pode também ser do tipo administrativa. Certos tipos de tráfego podem ser impedidos de atravessar enlaces específicos na rede, por medidas de segurança ou gerencial. Tais enlaces a serem excluídos seriam identificados por um atributo particular. O algoritmo então terá de computar uma rota que não inclua nenhum dos enlaces com o atributo particular. Da mesma maneira, é possível estabelecer restrições de modo que certo tipo de tráfego flua somente por certo conjunto de enlaces. Finalmente, restrições de desempenho podem ser combinadas com restrições de natureza administrativa.

Os algoritmos convencionais de roteamento IP não contemplam restrições. Basicamente, o roteamento baseado em restrições exige o cômputo da rota na origem. As restrições para um mesmo destino podem variar conforme as diferentes origens, e as restrições associadas a uma determinada origem (ou roteador de origem) são conhecidas somente por aquele roteador de origem, e não por outros roteadores na rede. No roteamento convencional IP, as rotas são calculadas de uma maneira distribuída por todos os roteadores da rede, e não consideram restrições de diferentes origens (mesmo porque, estas restrições são locais, não estando distribuídas por todos os roteadores da rede).

Outro motivo pelo qual o roteamento convencional IP não provê restrições é o fato de que o roteamento por restrições requer rotas explícitas (*source routing*) ou rotas definidas pela origem. O roteamento convencional IP usa o paradigma de encaminhamento baseado no destino. Ainda que o IP contenha um mecanismo de rota explícita, através da utilização do campo

options no pacote IP para *strict source routing* ou *loose source routing*, este não é suficiente para os objetivos do roteamento por restrições.

Mais ainda, uma vez que o cálculo da rota na origem deve considerar os dados dos atributos associados a cada enlace particular na rede, é necessário distribuir estas informações pela rede. O protocolo OSPF, por exemplo, somente distribui informações acerca do estado e métrica do enlace, e o protocolo RIP (protocolo de vetor de distância) distribui somente o endereço do próximo nó e sua distância. Assim, há que se estender ou criar um novo protocolo que implemente mecanismos a fim de distribuir informações acerca dos atributos extras associados aos enlaces (DAVIE; REKHTER, 2000). Tal foi feito com o próprio OSPF, resultando no protocolo estendido OSPF-TE (KATZ, 2003). Em conjunto, usa-se o algoritmo CSPF[1] (*Constraint OSPF*) (MARTINS, 2004b); este algoritmo é responsável por calcular as rotas, da mesma maneira que o OSPF, mas também considerando as restrições.

3.1. Componentes do Roteamento Baseado em Restrições

Um algoritmo ou protocolo de roteamento baseado em restrições deve, portanto, possuir um número de funções adicionais de modo a funcionar a contento.

A habilidade de computar uma rota na origem de modo a minimizar uma determinada métrica e considerando restrições (que é a

[1] Uma descrição do CSPF pode ser vislumbrada em http://en.wikipedia.org/wiki/Constrained_Shortest_Path_First.

primeira função adicional) requer que a origem tenha acesso a todas as informações necessárias a tal cálculo. Estas informações estão em parte disponíveis localmente e em parte residentes em outros roteadores na rede. Estas informações reúnem as várias restrições dos vários caminhos originados daquele roteador, bem como dados sobre a topologia da rede e atributos associados aos enlaces. Assim, a segunda função adicional é a capacidade de distribuir estas informações sobre topologia e atributos pela rede.

O encaminhamento, uma vez definida a rota, requer que os pacotes sigam a rota pré-determinada. Assim, a terceira função adicional é dar suporte a roteamento explícito (*source routing* ou *explicit routing*).

A quarta função adicional é a capacidade de reservar recursos e alterar os valores dos atributos associados aos enlaces, como resultado do uso de certas características dos enlaces pelos tráfegos de rede. Por exemplo, se a largura de banda mínima disponível for uma das restrições usadas para o estabelecimento da rota, o estabelecimento desta rota deve, pois, reduzir o valor da largura de banda total disponível nos enlaces utilizados (e também reservar a largura de banda para o tráfego em questão). A redução do valor da largura de banda disponível significa *alterar* o valor dos atributos associados à largura de banda nos enlaces utilizados, e, portanto, o algoritmo ou protocolo de roteamento baseado em restrições deve poder fazê-lo.

O protocolo de sinalização RSVP com as extensões de engenharia de tráfego, conhecido simplesmente como RSVP-TE (AWDUCHE *et al*, 2001), é capaz de suprir todas as funções requeridas, com exceção do cômputo de rotas baseadas num algoritmo CSPF. Como o RSVP-TE é um protocolo de sinalização, ele não se presta a calcular roteamento, que deve ser feito, por exemplo, pelo protocolo OSPF-TE.

3.2. Resumo do Capítulo

O roteamento baseado em restrições foi tema deste Capítulo. O roteamento convencional usa métricas como número de saltos (*hops*) ou nodos, ou ainda, valores definidos administrativamente. Os algoritmos de escolha da melhor rota procuram, então, minimizar tais métricas. No roteamento por restrições, além de minimizar uma métrica, a rota escolhida precisa obedecer a restrições ou requisitos definidos pelo administrador da rede. A rota, então, não pode violar tais restrições, que podem ser, por exemplo, certa largura de banda mínima disponível, um tamanho máximo de pacote, ou uma taxa máxima de pico de transmissão. A capacidade de dar suporte a rotas explícitas é característica fundamental do protocolo usado para roteamento por restrições.

4. Avaliação de Simuladores de Redes de Computadores Existentes

Um número de simuladores disponíveis no mercado foi apreciado no tocante às suas características, facilidade de uso, curva de aprendizado, documentação e conteúdo dos recursos requeridos. O objetivo foi verificar como cada simulador adequava-se a um estudo de impacto no desempenho de falhas de enlace numa aplicação VoIP (a existência dos protocolos necessários, a velocidade de aprendizado, a disponibilidade de documentação, o acesso livre ou gratuito à ferramenta). Basicamente, buscavam-se as seguintes características desejáveis num simulador, para os fins dos estudos desta Dissertação:

- Flexibilidade;
- Facilidade de configuração;
- Facilidade de implementação de novos recursos ou alteração do comportamento de recursos já existentes (personalização);
- Código livre e aberto (e gratuito);

- Uso direto, com o menor tempo de aprendizado inicial possível;
- Geração de resultados e estatísticas abundantes, de forma configurável e flexível, com o mínimo de pós-processamento;
- Existência das funcionalidades do MPLS, RSVP-TE e recuperação de falhas;
- Uso de uma linguagem de programação (preferencialmente o C) ou configuração, e não várias.

Notar aqui a preferência pela linguagem de programação C. Esta se justifica por ser o C uma linguagem que combina elementos de linguagens de alto nível com o controle e flexibilidade proporcionados por linguagens de máquina. Códigos escritos em C são geralmente bastante compactos e podem ser executados em uma grande variedade de sistemas computacionais diferentes. Os códigos de simulações preparados em C, com mínima ou nenhuma alteração, podem ser portanto executados em computadores dotados de sistemas operacionais diferentes (bastando para isso, logicamente haver um compilador adequado em cada sistema).

Os simuladores examinados foram: OPNET, NS-2, CSIM19, Cnet v.2.0.10, J-SIM e OMNET++. As análises serão detalhadas em cada subseção adiante, bem como as nuances de cada um que motivaram o desenvolvimento do protótipo próprio.

4.1. OPNET

O OPNET (OPNET, 2006), produzido pela empresa Opnet Technologies, Inc., foi o primeiro simulador estudado. Ele provê diversos módulos para simulação de redes, incluindo um vasto universo de protocolos e elementos de redes. O módulo para MPLS e RSVP-TE é vendido separadamente da versão padrão.

O OPNET é um produto comercial e a Universidade para a qual este Autor submete esta Dissertação não dispõe de uma licença de uso. Há uma versão acadêmica, gratuita, mas esta versão tem recursos limitados e sua documentação é pobre. Desta forma, este foi um detalhe motivador para o desenvolvimento do protótipo.

4.2. NS-2

O NS-2 – *Network Simulator* – (NS-2, 2006) é um simulador de eventos discretos orientado a pesquisas em redes de computadores. É de código livre, desenvolvido principalmente pelo projeto VINT[1] (*Virtual InterNetwork Testbed*), pelo laboratório PARC[2] (*Palo Alto Research Center*) da Xerox, UCB[3] (*University of California at Berkeley*), USC/ISI[4] (*University of South California – Information Sciences Institute*), e também conta com contribuições vindas de vários pesquisadores e usuários.

[1] http://www.isi.edu/nsnam/vint
[2] http://www.parc.xerox.com
[3] http://www.berkeley.edu
[4] http://www.isi.edu

O NS-2 é codificado em C++ em uma estrutura modular. O usuário interage com o simulador através da linguagem do tipo *script* OTcl, orientada a objetos. Foi concebido para ser executado nativamente em sistemas Unix, incluindo o Linux, apesar de ser possível sua instalação e uso no Microsoft Windows (PORTNOI; ARAUJO, 2002).

As funcionalidades dos procotolos MPLS e RSVP-TE não estão disponíveis como bibliotecas padrão no NS-2. Foram implementadas na forma de contribuição de terceiros e ofertadas de maneira não centralizada, com cada pesquisador colocando sua contribuição numa página *web*, usando ou não implementações feitas por outros. O módulo MNS (MPLS *for Network Simulator*), que possui funcionalidades do MPLS, foi desenvolvido por (GAEIL, 199-?; GAEIL; WOOJIK, 2000, 2001), mas sua localização original não mais está acessível na internet. Além do MPLS, este módulo também inclui o CR-LDP, mas não o RSVP-TE. O módulo MNS foi posteriormente incrementado por (BOERINGER, 2006) e (ADAMI, 2005; CALLEGARI; VITUCCI 2006) para oferecer recursos do RSVP-TE. Estes códigos não podem ser obtidos diretamente dos sítios dos autores na internet, mas somente através de um pedido via e-mail ou de outros usuários que já possuam os módulos.

A curva de aprendizado do NS-2 é significativamente íngreme. O usuário deve dominar a linguagem OTcl e saber usá-la para construir programas que interagem com os objetos do simulador, estes codificados em C++. A documentação existente não é escrita em estilo didático, o que torna difícil para um principiante montar simulações iniciais, sem antes investir

considerável tempo em tentativas e erros. A documentação é especialmente fraca para os módulos MPLS e RSVP-TE, o que requer do usuário a leitura do código-fonte destes módulos de forma a aprender como interagir com eles e conhecer todos os recursos oferecidos. O código do simulador é livre, mas, novamente, programar novas funções ou modificações demanda o estudo de largas porções do código-fonte (tanto em C++, como em OTcl), distribuído em centenas de arquivos individuais.

A geração de resultados e estatísticas não é automática, ao final de uma simulação. O usuário deve gerar um arquivo de *trace*[1] durante a simulação e então, após esta finalizada, realizar um pós-processamento no arquivo de *trace* (com o uso de uma linguagem de processamento adequada, como o *awk*), de maneira a calcular as estatísticas desejadas. Simulações podem facilmente gerar arquivos de *trace* muito grandes, o que demanda um tempo de pós-processamento apreciável.

O suporte deficiente do MNS, que é um módulo necessário para as simulações desejadas nesta Dissertação – e a consequente dificuldade em alterar ou programar novas funções possivelmente requeridas no NS-2 – serviu como incentivo para a confecção do novo protótipo de simulador.

---

[1] *Trace*, ou rastreamento, é o processo pelo qual um programa registra resultados selecionados, de forma contínua, iterativamente, em algum meio escolhido (tipicamente em um arquivo de texto). Por exemplo, um simulador de redes pode registrar em um arquivo cada novo evento gerado, como, por exemplo, um pacote sendo recebido em um nó.

4.3. CSIM19

O CSIM19 (MESQUITE, 2006) é um simulador orientado a processo, baseado em eventos discretos, disponível tanto em C, como em C++ e Java. É um conjunto de bibliotecas e funções que um programa (escrito na mesma linguagem das bibliotecas) pode dispor de sorte a modelar um sistema e simulá-lo. O CSIM19 é um simulador de uso geral, não específico para redes de computadores, e é comercial. Não há uma versão gratuita ou livre; o código livre e gratuito é, novamente, condição desejada pelo Autor.

4.4. Cnet v.2.0.10

Desenvolvido principalmente para uso em cursos de graduação de redes de computadores, o Cnet (McDONALD, 2006) é um simulador orientado a eventos escrito em C. Ele faz uso das linguagens Tcl/Tk para implementar sua interface gráfica, por intermédio da qual o simulador mostra uma representação da topologia modelada e permite que alguns atributos sejam configurados. As topologias, por sua vez, são construídas através de arquivos especiais.

O objetivo do simulador, segundo seu autor, é permitir a experimentação com protocolos de rede. Ele traz, nativamente, o IEEE 802.3 (Ethernet) e WAN (*Wide Area Network*) ponto-a-ponto, e também mecanismos para causar corrupção ou perda de quadros de dados probabilisticamente.

O manual do Cnet declara que o simulador somente funciona sob plataformas Unix/Linux. Não é compatível, portanto, com sistemas Microsoft Windows ou Apple Macintosh e também não inclui os protocolos MPLS e RSVP-TE. Apesar de que o código-fonte da ferramenta é livre, não é extensivamente documentado. A geração de estatísticas pós-simulação é apenas básica. Como o foco do simulador é a construção e experimentação de protocolos, e não análise de desempenho e QoS, conjuntamente com suas limitações de plataforma e ausência do MPLS e RSVP-TE, a análise do Cnet também incitou a produção do protótipo de ferramenta de simulação.

4.5. J-SIM

Há um número de simuladores batizados J-Sim disponíveis através de uma simples busca na internet. O J-Sim aqui descrito (HUNG-YING, 2006) é de código livre, baseado em componentes e escrito em Java. Inclui o MPLS através de uma extensão codificada por terceiros (JAVASIM, 2006), mas não inclui o protocolo de sinalização RSVP-TE. A documentação está presente no sítio do simulador na *web*, e de fato apresenta boas descrições do código-fonte, a filosofia por trás do simulador e alguns tutoriais e guias para novas implementações.

A instalação do simulador no computador de trabalho, como parece ser usual com aplicações Java, requer a configuração de variáveis de ambiente, a compilação dos códigos-fonte com auxílio de ferramentas mais comuns em plataformas Linux, e então a aplicação de correções (*patches*) necessárias para as extensões (como a extensão MPLS). O J-Sim é um

ambiente com duas linguagens: o usuário manipula classes escritas em Java usando *scripts* codificados em Tcl, de maneira muito similar ao NS-2. Esta característica resulta nos mesmos problemas verificados no NS-2, i.e., a exigência de se conhecer e manipular duas linguagens de sorte a usar o simulador e implementar funcionalidades não existentes. Da mesma forma, incentivou a montagem do protótipo de simulador.

4.6. OMNET++

Este simulador (VARGA, 2006) é um ambiente de eventos discretos programado em C++, composto de módulos ou componentes que são por sua vez montados em modelos, através de sua linguagem interna chamada NED (*NEtwork Description*). Criado primariamente para simulações de redes de computadores, seu autor informa que esta ferramenta aceita simulações também de redes de filas e outros sistemas.

Uma simulação de redes de computadores é construída a partir de um modelo que representa as funcionalidades e ações de uma rede, denominado INET *Framework*. Este modelo é encontrado no sítio do simulador na internet, assim como vários outros modelos para simulação de diferentes sistemas. O INET *Framework* foi primeiramente programado por Xuan Thang Nguyen, mas o sítio original não mais está acessível na internet. Aparentemente, este modelo inclui os protocolos MPLS, LDP e RSVP-TE. Não está claro, e a extensa documentação não o menciona, se o componente RSVP-TE inclui a recuperação rápida ou recuperação de falhas.

O uso do OMNET++ não é simples. Ele possui várias janelas e menus, resultando numa interface carregada (poluída, de certa forma) e requerendo consultas à extensa documentação colocada na internet, a tutoriais e demonstrações antes que se possa começar a construir topologias de simulação. Apesar de ser promissor, optou-se pela construção do protótipo de simulador, motivado pela característica de ser o OMNET++ codificado em C++ e ainda utilizar uma segunda linguagem interna (a NED) para montar as topologias, e também pelo fato de não estar claro se este simulador provê recuperação de falhas no RSVP-TE, o que obrigaria uma análise de seu vastíssimo código-fonte.

4.7. Decisão sobre a Ferramenta de Simulação a Adotar

Cada simulador analisado demonstrou características interessantes, especialmente o NS-2, que é um dos mais extensivamente usados em pesquisa, e o OMNET++, que exibe poder com sua bela interface gráfica. Nenhum deles, contudo, exibiu o conjunto completo de funcionalidades desejado para as pesquisas do Autor, relacionadas no início deste Capítulo.

A decisão sobre a ferramenta de simulação a adotar, portanto, tomando como motivador a investigação feita, recaiu sobre a construção de um simulador próprio que reunisse os recursos desejados, na linguagem de

programação C. Este simulador foi batizado de TARVOS[1] *Computer Networks Simulator* (PORTNOI; MARTINS, 2007).

4.8. Resumo do Capítulo

Do ponto de vista do estudo (através de simulação) do desempenho de uma aplicação VoIP, submetida a uma falha de enlace sobre uma rede MPLS, necessita-se de uma ferramenta de simulação que contenha uma série de características, dentre elas: prover os protocolos MPLS e RSVP-TE, conter o mecanismo de recuperação rápida de falhas do RSVP-TE, possuir flexibilidade na geração de estatísticas e resultados rápidos, evitar um uso demasiado complicado e poder ser configurado e modificado, livremente.

Dos simuladores mais diretamente disponíveis, foram examinados o OPNET, NS-2, CSIM19, Cnet v.2.0.10, J-SIM e OMNET++. As qualidades e deficiências de cada um foram levantadas, culminando com a conclusão de que nenhum deles reunia o conjunto completo de características desejadas para o estudo. Além disso, a inclusão destas características em alguns deles esbarraria na principal dificuldade, que é a leitura, estudo e compreensão de grande parte do código-fonte destes simuladores, às vezes em linguagem de programação sem as distintivas desejáveis para a utilização na pesquisa proposta: portabilidade (facilidade de levar o código, com mínima ou nenhuma alteração, para sistemas computacionais diferentes), compacidade do código, flexibilidade e controle.

---

[1] *Tarvos* é uma palavra gaulesa (uma língua céltica) que significa touro. Da mitologia gaulesa, *Tarvos Trigaranus* era um deus touro com três gruas pousadas em suas costas (*Garanus* é a palavra gaulesa para grua). O nome foi usado para batizar uma das luas do planeta Saturno, descoberta em 23 de setembro de 2000, e também o presente simulador.

O Capítulo traz a conclusão de que a opção feita foi conceber um simulador próprio, o TARVOS *Computer Networks Simulator*.

5. Construção do TARVOS.

Para os intentos da pesquisa, há a necessidade de usar uma ferramenta de simulação que reúna um número de características, quais sejam:

- Configurável e personalizável, de modo que as características dos protocolos, geradores de tráfego, enlaces e outros elementos de rede possam ser livremente modificados, estendidas, controladas ou desabilitadas.
- Flexível nas configurações, de forma que comportamentos diferentes possam ser mais facilmente simulados;
- De uso direto e pragmático, ou seja, evitando gasto de tempo em aprendizado de inúmeros recursos talvez não utilizados ou interfaces muito complexas, com diversas linguagens;
- De código livre, aberto e gratuito, permitindo assim sua modificação e uso sem necessidade de licença;

- Geração de resultados rápidos com diversos tipos de estatísticas de interesse para os estudos de desempenho em redes de computadores, em formatos práticos para o usuário (em arquivos que possam ser exportados para planilhas, por exemplo) e com o mínimo de pós-processamento. Preferencialmente, que os resultados e estatísticas já sejam gerados em tempo de simulação no formato desejado para os estudos;
- Existência das funcionalidades dos protocolos RSVP-TE e MPLS;
- Para o contexto do estudo do caso específico abordado nesta Dissertação, a recuperação rápida em caso de falha;
- Uso da linguagem de programação C, de modo a privilegiar portabilidade, compacidade, flexibilidade do código.

De sorte a reunir todas estas características, decidiu-se por construir uma nova ferramenta de simulação própria, batizada de TARVOS *Computer Networks Simulator*, ou simplesmente TARVOS. Além de buscar cada uma das características desejadas, este simulador também foi codificado com uso extensivo de comentários no código-fonte a fim de compor uma documentação prática. Não obstante, pretende-se aqui também documentá-lo, procurando um norte didático, e oferecê-lo à comunidade sob formato de código livre. A contribuição intentada é a apresentação de um simulador de redes de computadores (que, em adição,

também se presta à simulação de sistemas de redes de filas) que exibe as características descritas anteriormente e que seja de uso prático, acessorado por uma documentação didática e de código-fonte razoavelmente pouco extenso.

As subseções adiante descreverão o simulador em seus pormenores.

5.1. Características

TARVOS é um simulador[1] (ou ambiente de simulação, ou ferramenta de simulação) orientado a eventos discretos, codificado inteiramente em linguagem C. Foi desenvolvido como uma extensão para programas em C; é composto de várias funções e estruturas de dados que modelam sistemas de filas e, em cima destes, elementos de redes de computadores. As mensagens geradas pelo simulador e os nomes de funções, estruturas e variáveis foram mantidas em inglês, pois esta é a língua primordial para intercâmbio de ferramentas de simulação.

O TARVOS consiste em três elementos principais: o *kernel* ou núcleo, o *shell 1* e o *shell 2*, conforme demonstrado na Figura 2.

[1] O Autor usa aqui "simulador" numa acepção generalista, sem diferenciar "simulador" de "ambiente de simulação" ou de "ferramenta de simulação". Compreende-se que, sem um modelo (seja este modelo confeccionado através de uma interface gráfica ou através de um programa em C ou Tcl), um simulador, ou ambiente de simulação, ou ferramenta de simulação nada fazem.

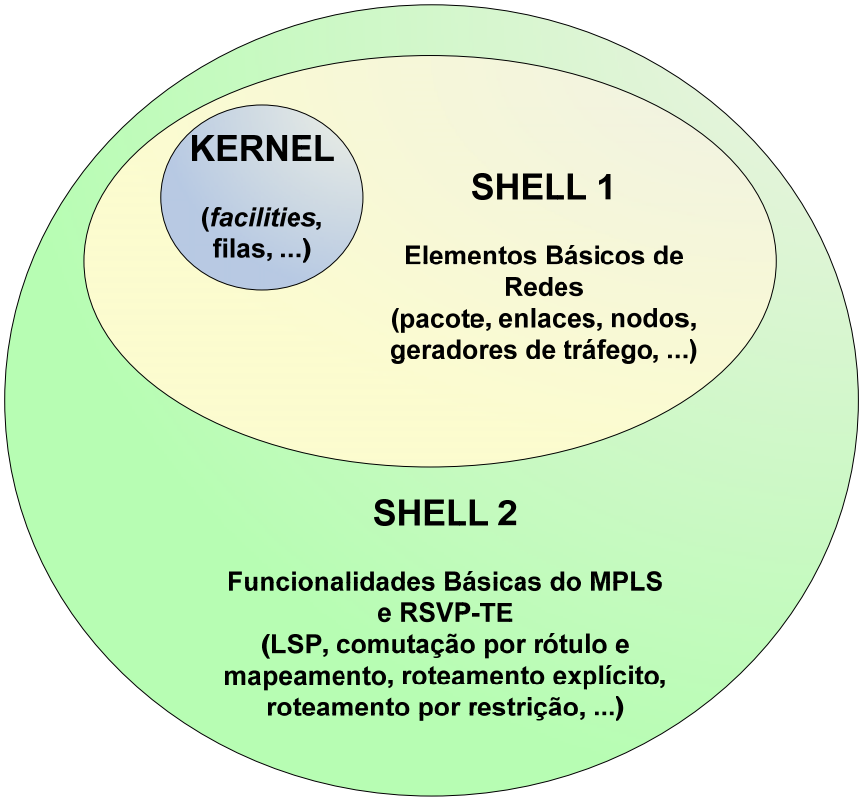

Figura 2: Estrutura do simulador TARVOS em seus três elementos principais.

5.1.1. O *Kernel*

O *kernel* ou núcleo do TARVOS é um simulador de sistemas de filas ou redes de filas, de eventos discretos, baseado no simulador SMPL (MacDOUGALL, 1987). O autor do código original[1] deste *kernel*, Sérgio de Figueiredo Brito (2002), da Universidade Salvador, concebeu-o reescrevendo o SMPL com o uso de alocação dinâmica, estruturas de dados e ponteiros (onde o SMPL restringe-se ao uso de matrizes e vetores objetivando a economia drástica de memória, necessária na época de sua criação). O novo código também permite que uma estrutura de dados complexa possa ser passada através dos eventos, onde o SMPL original consentia apenas a passagem de um inteiro simples.

[1] Que foi batizado *SimM – Simulation Machine.*

O *kernel* dispõe de elementos básicos de sistemas de filas, como recursos e servidores que provêm os serviços; filas FIFO (*First In, First Out*) ou FCFS (*First Come, First Served*) – onde o primeiro a entrar na fila será o primeiro a ser atendido – e filas de prioridade (PQ – *Priority Queue*) para os recursos e servidores. Também, funções e estruturas de dados para manipulação dos eventos, e funções estatísticas, geradores de números aleatórios e geradores de variáveis aleatórias (baseadas em distribuições de probabilidade, como exponencial e uniforme). Este Autor modificou e estendeu o *kernel* original de forma a que este proveja servidores inoperantes (ou *down*), a correção do comportamento da fila PQ, mais funções estatísticas e mais alguns geradores de variáveis aleatórias, como as baseadas na distribuição de Pareto e a exponencial *On/Off*.

O *kernel* enxerga um sistema como uma combinação de três componentes essenciais ou entidades: **recursos** (ou *facilities*), ***tokens*** (ou fichas) e **eventos** (Figura 3).

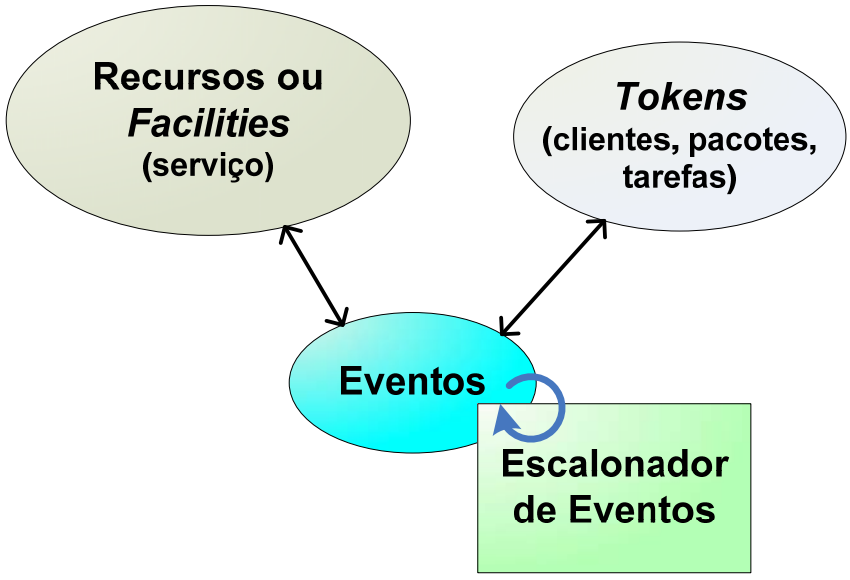

Figura 3: Relacionamento entre as entidades de um sistema, como enxerga o *kernel* do TARVOS.

5.1.1.1. Recursos ou Facilities

Um sistema é composto por uma coleção interconectada de recursos (MacDOUGALL, 1987). Estes recursos, no mundo real, podem ser roteadores, processadores, caixas de banco ou servidores de *web*, por exemplo. Basicamente, qualquer coisa que presta algum tipo de *serviço* é um recurso. Os recursos são chamados de *facilities* no TARVOS, por herança da denominação nativa do SMPL e também porque a função no TARVOS que primeiro cria uma *facility* é exatamente assim denominada. Doravante, pois, neste documento, usar-se-á esta nomenclatura em inglês a fim de permitir uma relação imediata entre idéia e comando do simulador. As *facilities* possuem uma série de funções a elas relacionadas: **definição**, **requisição de serviço** ou **reserva**, **liberação**, **preempção** e **estado** (ou *status*).

Uma *facility* é *definida* através da configuração de seu nome e o número de servidores que a compõem. Quando um cliente requer um serviço, este serviço é "desempenhado" de forma simulada (ver Seção 5.1.1.3) por uma determinada *facility*. Então, a *facility* é sinalizada na forma de uma *reserva*, que lhe é solicitada para o cliente em questão. Quando o serviço é concluído, a *facility* é *liberada*. A condição ou estado de ocupada ou livre de uma *facility* é reportada pelo seu *estado* ou *status*.

5.1.1.2. Os Tokens

Os *tokens* ou fichas representam as entidades ativas do sistema. Os *tokens* podem modelar clientes, tarefas, pacotes de rede, pessoas, automóveis, bytes, etc. O comportamento dinâmico de um sistema é

modelado pelo fluxo de *tokens* através da coleção de *facilities* ou recursos. No TARVOS, um *token* é uma estrutura de dados (uma *struct*). Se, por exemplo, o *token* deve modelar um pacote de rede, então a estrutura de dados conterá informações típicas de um pacote, como nodo de origem, nodo de destino, número ID do pacote, rótulo MPLS, prioridade, etc.

5.1.1.3. Relação entre Tokens e Facilities

As *facilities*, portanto, "executam" um serviço nos *tokens*. Destaca-se aqui o termo "executam" entre aspas, pois, no simulador, nenhum serviço é efetivamente executado, mas sim *simulado*; para a avaliação de desempenho, o *tempo* de execução do serviço é relevante, e não o serviço em si. O simulador, por conseguinte, simula o serviço retendo o *token* no servidor (ou seja, criando um atraso no *token*) pelo tempo apropriado.

Um roteador típico, modelado como uma *facility* com um servidor, desempenharia tarefas como receber, transmitir ou enfileirar um pacote (este, por sua vez, modelado como uma *token*). Um enlace de rede, assim, é representado como uma *facility* capaz de transmitir um pacote a uma determinada velocidade (calculada a partir da largura de banda do enlace), e então propagar este pacote através do meio de comunicação. A propagação em si é modelada como um atraso constante (ou um servidor com tempo de serviço fixo e capacidade infinita), cujo valor depende do tipo de meio pelo qual trafegam os datagramas, ou seja, o material construtivo do enlace e a distância entre os nós interligados pelo enlace (ver Seção 5.1.2.2).

No processo de simulação, se um *token* requer ser servido por uma *facility*, então um pedido de *reserva* é feito. Se a *facility* possui servidores livres, o *token* é colocado em serviço e o tempo em que o serviço deve ser concluído é *escalonado*. Este instante de conclusão de serviço é chamado *evento*. No instante escalonado, o servidor, dentro da *facility*, que manteve o *token* em serviço é então *liberado*. O *token* deve prosseguir seu caminho através do sistema, solicitando reservas em outras *facilities*.

Se, ao requisitar serviço, um *token* depara-se com uma *facility* com todos os seus servidores ocupados, então este *token* é *enfileirado* na fila da *facility*. O TARVOS provê uma fila de prioridades – PQ – para cada *facility*. As prioridades são representadas por um número inteiro, onde números mais altos significam maior prioridade. A prioridade é especificada quando da requisição de serviço (ou reserva) para uma *facility*, e pode estar contida dentro da estrutura de dados do *token*. Quando a *facility* é liberada, ela coleta o primeiro *token* na cabeça de sua fila e coloca-o em serviço.

É interessante mencionar que o tempo de serviço para um *token* é especificado quando o pedido de reserva ou requisição de serviço para ele é feito para uma *facility*. Se o *token* é enfileirado (por não haver servidores livres), este tempo de serviço é gravado; quando o *token* é desenfileirado e colocado em serviço, o simulador recupera o tempo de serviço gravado para ele e escalona o final de serviço, ou seja, a liberação da *facility*.

Uma *facility* também pode sofrer *preempção*. Quando um *token*, com uma determinada prioridade, requisita serviço e a *facility* está ocupada,

então um *token* em serviço naquela *facility*, que tenha uma prioridade menor que o *token* solicitante, terá seu serviço interrompido. O tempo restante de serviço será computado e este *token* interrompido será enfileirado à frente de outros *tokens* de mesma prioridade ou menor. O *token* requisitante com maior prioridade tomará então seu lugar no servidor agora livre. Quando este *token* terminar seu serviço, o servidor é liberado e o simulador desenfileira o *token* na cabeça da fila. Se este *token* na cabeça da fila for o *token* que sofreu a preempção, então o servidor reservado para ele terá seu tempo de liberação (ou final de serviço) escalonado para o restante do tempo de serviço computado e gravado para o *token* interrompido.

Se, quando requisitando uma preempção numa *facility* ocupada, nenhum *token* com prioridade menor que o *token* solicitante for encontrado, então o *token* solicitante é enfileirado da mesma forma que uma requisição de serviço normal, não preemptiva.

5.1.1.4. Os Eventos

Um *evento* é qualquer mudança de estado no sistema. Por exemplo, a chegada de um novo *token*, uma requisição de serviço para um *token*, a liberação de uma *facility*, um desenfileiramento. O simulador identifica os eventos por um número inteiro, o instante de tempo quando este evento deve ocorrer e o *token* relacionado a ele. O *escalonador de eventos* gerencia os eventos organizando-os numa *cadeia de eventos*, que é uma lista duplamente encadeada, ordenada ascendentemente por tempo.

Eventos são continuamente gerados na simulação, até que um critério de parada seja cumprido. Três funções principais estão relacionadas com eventos: *escalonar* um evento (ou seja, colocar um evento na cadeia de eventos); *causar* um evento (retirar o próximo evento da cadeia de eventos); e *cancelar* um evento (removê-lo da cadeia de eventos e deletá-lo).

5.1.1.5. Relação entre Eventos e Tokens

Os *tokens* movem-se entre as *facilities* através do escalonamento de eventos no tempo. A interconexão entre as *facilities* não é explícita para o *kernel* do TARVOS. Esta interconexão é derivada, implicitamente, do roteamento dos *tokens* por entre as *facilities*. O roteamento, por seu turno, é definido pelo processamento de cada tipo de evento.

Por exemplo, seja um sistema hipotético chamado “digestão”, composto de quatro *facilities*: “mastigação”, “deglutição”, “estômago” e “intestino” (Figura 4).

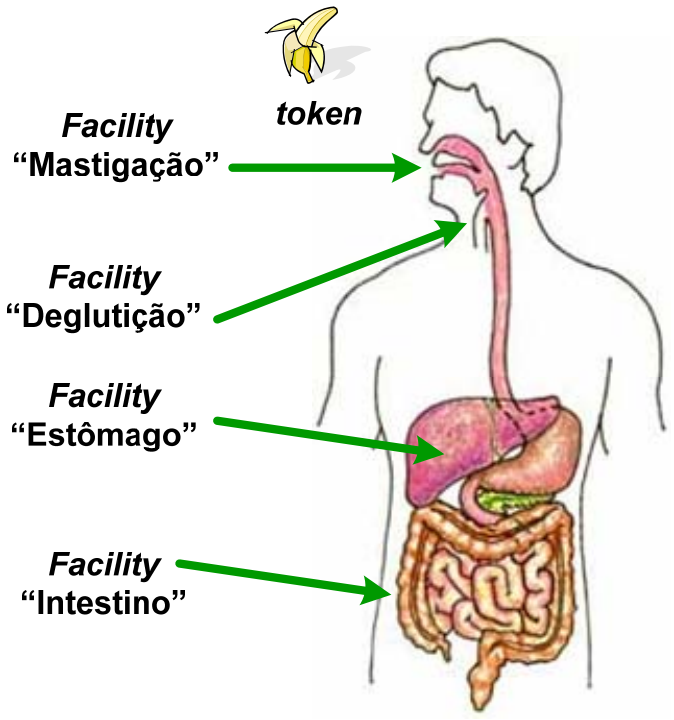

Figura 4: Sistema hipotético "digestão", composto de quatro *facilities*.

O *token* ou cliente deste sistema é um naco de comida, que passa pelas *facilities* e recebe delas serviços: ser mastigado, engolido, receber sucos gástricos e ser absorvido ou eliminado pelo intestino.

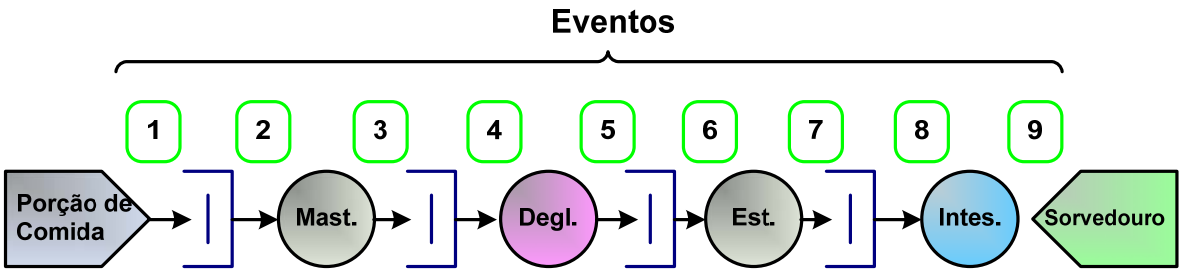

Figura 5: Diagrama em rede de filas do sistema "digestão".

A Figura 5 traz a representação em rede de filas do sistema exemplificado. Vê-se que o *token* atravessa, em seqüência, todas as *facilities* até atingir o sorvedouro (que é o final do sistema). Identificados em quadrados através dos números de 1 a 9, estão os eventos individuais do

sistema. Sob o ponto de vista do simulador, estes eventos devem ser processados conforme a Tabela 1.

Tabela 1: Relação de eventos e seus tratamentos do sistema "digestão".

Número do Evento	Descrição e Procedimentos
1	Chegada de um *token* ao sistema. Requisitar serviço na *facility* "mastigação", através do escalonamento do evento 2. Programar nova chegada de outro *token*, através do escalonamento do evento 1.
2	Reserva da *facility* "mastigação". Se *facility* estiver livre (ou seja, se a boca estiver vazia), escalonar término do serviço, que é o escalonamento do evento 3. Se estiver ocupada, enfileirar *token*.
3	Liberação da *facility* "mastigação". Encaminhar o *token* adiante, para a *facility* "deglutição". Isto é feito escalonando o evento 4.
4	Reserva da *facility* "deglutição". Se estiver livre, escalonar evento 5 (término de serviço). Senão, enfileirar.
5	Liberação da *facility* "deglutição". Encaminhar *token* adiante para "estômago", escalonando evento 6.
6	Reserva da *facility* "estômago". Se estiver livre, escalonar evento 7 (término de serviço). Senão, enfileirar.
7	Liberação da *facility* "estômago". Encaminhar *token* adiante para "intestino", escalonando evento 8.
8	Reserva da *facility* "intestino". Se estiver livre, escalonar evento 9. Senão, enfileirar.
9	Liberação da *facility* "intestino" e final/saída do sistema.

Observar que o roteamento dos *tokens* por entre as *facilities* é definido pelo processamento dos eventos. Um *token* que acabou de passar pela *facility* "mastigação" deve seguir para "deglutição". Isto está implícito nas ações do evento 3 (que é a liberação da *facility* "mastigação"), que escalona o evento 4 (pedido de reserva na "deglutição") após fazer a liberação da *facility*. Tipicamente, os escalonamentos de eventos de chegada de novo *token* ao sistema e liberação de *facility* têm o instante de ocorrência determinado segundo uma variável aleatória; os escalonamentos de eventos

do tipo reserva acontecem de imediato, ou seja, com um intervalo de tempo zero.

5.1.2. O *Shell* 1

O *shell 1* provê bibliotecas que implementam elementos fundamentais de rede, como pacotes, enlaces do tipo *simplex* e *duplex*, nodos, roteamento estático, falhas em enlaces e geradores de tráfego exponencial *On/Off* e CBR. Da mesma forma, compreende estruturas de dados que facilitam o cálculo de medidas de desempenho e estatísticas, e funções de geração de *traces*. As funções do *shell* 1 usam primordialmente funções do *kernel* para realizar seus processamentos.

5.1.2.1. Os Nodos

Os *nodos* são modelados como entidades capazes de receber pacotes dos enlaces, encaminhar pacotes para os enlaces, tomar decisões a respeito de caminhos e rotas, descartar pacotes e coletar estatísticas (como atraso e *jitter*). Em suma, os nodos comportam-se similarmente a roteadores.

5.1.2.2. Os Enlaces

As entidades *enlaces* conectam dois nodos em um só sentido (enlaces *simplex*) ou nos dois sentidos (enlaces *duplex*). O TARVOS modela os enlaces como *facilities* de um servidor por enlace *simplex*, seguidos por um centro de atraso, que insere um tempo de atraso fixo. Assim, um enlace *simplex* corresponde a uma *facility* de um servidor, e um enlace *duplex* é formado por uma *facility* com um servidor em um sentido da comunicação, e

por uma *facility* com um servidor no outro sentido da comunicação (sempre seguidos de um centro de atraso). O serviço prestado pelos enlaces é o transporte de pacotes de uma ponta do enlace para a outra ponta, i.e., de um nodo para o outro. Isto é feito em duas etapas: a *transmissão* (função executada pelo servidor) e a *propagação* (função executada pelo centro de atraso). Ver Figura 6. (O APÊNDICE B exibe os procedimentos usados de sorte a validar a modelagem do enlace.)

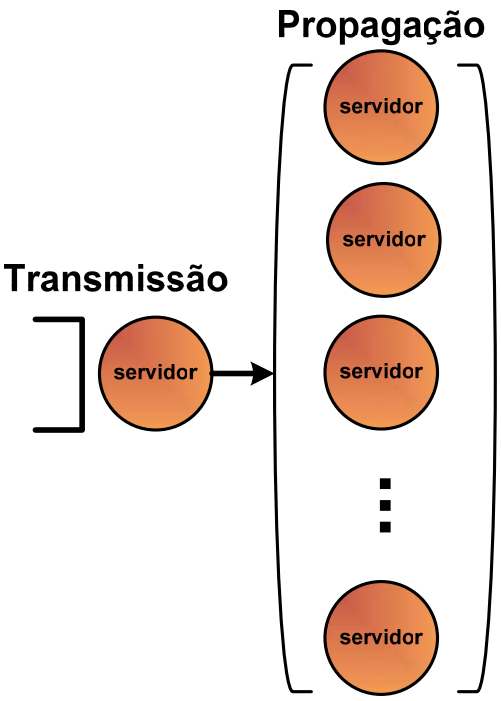

Figura 6: Modelagem em duas etapas (transmissão e propagação) de um enlace *simplex* no TARVOS.

Na *transmissão*, os enlaces transmitem o pacote à plena velocidade da largura de banda. O tempo em que o pacote leva para ser transmitido é igual ao tamanho do pacote dividido pela largura de banda. A fórmula é vista na Equação 1, onde $T_{transmissão}$ é o tempo de transmissão em segundos, *tam* é o tamanho do pacote em bytes, e *bw* é a largura de banda em bits por segundo.

$$T_{transmissão} = \frac{tam * 8}{bw} \quad (1)$$

Se a transmissão for bem sucedida, então o TARVOS agora considera que o pacote está “dentro” do meio de comunicação, ou seja, está “dentro” do fio em forma de sinais elétricos, ou no ar, em forma de ondas eletromagnéticas, ou ainda, em forma de luz numa fibra ótica, por exemplo. Tem-se então a segunda etapa, que é a *propagação* do pacote pelo meio até o nodo de destino do enlace. A propagação é um simples atraso de duração fixa, definido pelo usuário (ou seja, o centro de atraso ou servidor infinito da Teoria de Filas) (JAIN, 1991). Este tempo de atraso ou tempo de propagação é função do tipo de meio, ou seja, o tipo de material construtivo do enlace, a correspondente velocidade de propagação neste meio e o comprimento físico do enlace (a distância que separa os dois nós interligados pelo enlace). Deste modo, o tempo ou atraso de propagação é calculado segundo a Equação 2, onde *propag* é o tempo de propagação, d é o comprimento do enlace (distância entre os dois nós interligados pelo enlace) e v é a velocidade de propagação do meio.

$$propag = \frac{d}{v} \quad (2)$$

A Tabela 2 ilustra algumas velocidades de propagação aproximadas em meios mais usados em enlaces de redes de computadores. As velocidades exatas podem ser obtidas nas especificações fornecidas pelos

fabricantes dos meios utilizados pelos enlaces. Este nível de precisão, entretanto, provavelmente é desnecessário para a maioria das simulações. Como regra prática, costuma-se usar a velocidade de 2/3 da velocidade da luz para cabos e fibras óticas. Para um enlace via satélite geoestacionário, usa-se comumente o tempo total de propagação (estação-satélite-estação) de 240ms, que é a soma do tempo de propagação da estação origem até o satélite e do satélite até a estação destino[1].

Tabela 2: Velocidades de propagação em alguns meios (adaptado de http://www.wikifaq.com/Ethernet_FAQ).

Meio	Velocidade de Propagação (c = velocidade da luz no vácuo = 300.000 km/s)
Cabo coaxial grosso	0,77 c = 231.000 km/s
Cabo coaxial fino	0,65 c = 195.000 km/s
Cabo de par trançado	0,60 c a 0,70 c = 180.000 a 210.000 km/s
Fibra ótica	0,65 c = 195.000 km/s

Um enlace de rede (em um sentido ou *simplex*), por conseguinte, é modelado pelo TARVOS como sendo uma *facility* de um servidor, com uma fila PQ, conectada a um servidor de capacidade infinita (ou centro de atraso, que representa o meio). O tempo total que um pacote leva para ir de um nodo a outro, através de um enlace, é portanto definido pela Equação 3, onde $T_{nodo\text{-}nodo}$ é o tempo total de viagem de um pacote entre um nodo e outro do mesmo enlace, *tam* é o tamanho do pacote em bytes, *bw* é a largura de banda do enlace em bits por segundo, e *propag* é o tempo de propagação do meio em segundos.

[1] Um satélite geoestacionário orbita o planeta a uma altitude de cerca de 36.000km. O tempo de propagação do sinal entre uma estação e o satélite é, assim, aproximadamente, 36.000km / 300.000km.s^{-1} = 120ms. O tempo total de propagação entre duas estações (origem-satélite-destino) será de 120 + 120 = 240ms.

$$T_{nodo-nodo} = \frac{tam * 8}{bw} + propag \quad (3)$$

Um enlace tem, associado a si, em cada sentido, uma quantidade de recursos ou haveres disponíveis. Estes recursos ou haveres são o CIR (*Committed Information Rate*), o PIR (*Peak Information Rate*) e o CBS (*Committed Bucket Size*), que serão definidos adiante, na Seção 5.1.3.1. Os recursos podem ser reservados para determinados fluxos de tráfego, de modo que tenham uma garantia mínima de qualidade. Ao haver uma reserva, os recursos disponíveis no enlace diminuem; ao serem devolvidos, os recursos disponíveis aumentam. As reservas são tratadas na Seção 5.1.3.3.

O recurso CBS é definido pelo usuário. Os recursos PIR e CIR são calculados, na configuração do enlace, para o valor da largura de banda dividido por oito (pois a largura de banda é definida em bits por segundo, e os valores PIR e CIR têm, no TARVOS, bytes por segundo como unidade).

5.1.2.3. Os Pacotes

Como mencionado anteriormente, os pacotes são estruturas de dados para o simulador. Várias informações são incorporadas na estrutura do pacote: tamanho do pacote em bytes; nodo origem e nodo destino; número ID ou identificador único do pacote; rótulo MPLS; ID de mensagem de controle; tipo de mensagem de controle; objeto rota explícita e TTL (*Time To Live*) são algumas destas informações.

5.1.2.4. Os Geradores de Tráfego

Os *Geradores de Tráfego* são entidades virtualmente conectadas aos nodos, que geram pacotes de acordo com distribuições de probabilidade. Estas distribuições de probabilidade, por sua vez, modelam estocasticamente tráfego real de rede. O TARVOS inclui geradores CBR, Exponencial ou Poisson, Exponencial *On/Off* e Pareto. O gerador exponencial *On/Off* é especialmente útil para modelar tráfego gerado por aplicações VoIP.

O modo como o tráfego gerado é transportado, no TARVOS, é uma simulação do comportamento do protocolo UDP (*User Datagram Protocol*) sobre IP. Os controles de fluxo e congestionamento do protocolo TCP não estão programados.

5.1.3. O *Shell* 2

O *shell 2* consiste em bibliotecas que provêm as funcionalidades básicas do plano de controle MPLS e RSVP-TE, bem como encaminhamento e sinalização, comutação por rótulo, criação de LSPs (*Label Switched Paths*) primárias e *backup*, roteamento explícito e baseado em restrições, manutenção do *soft-state* do RSVP, policiador de tráfego (*policer*), mensagens de controle PATH, RESV e HELLO, e mecanismos de detecção e recuperação de falhas. O *shell 2* usa as funções do *kernel* e *shell 1* para criar os seus recursos.

5.1.3.1. O Policiador (Policer)

Antes de ser efetivamente transmitido, os pacotes podem ser submetidos ao *policiador* (*policer*), que é baseado num algoritmo *Token*

Bucket (TANENBAUM, 1996). O *token bucket*, no TARVOS, trabalha efetivamente com os seguintes parâmetros:

- CIR (*Committed Information Rate*) – taxa em bytes por segundo de enchimento do *bucket*;
- CBS (*Committed Bucket Size*) – tamanho máximo do *bucket* em bytes;
- PIR (*Peak Information Rate*) – taxa máxima em bytes por segundo (este parâmetro não é utilizado pelo policiador na atual versão outubro/2007 do protótipo);
- CurrentTime – tempo ou instante atual;
- LastTime – último instante de tempo em que o *bucket* foi usado;
- cBucket – tamanho atual do *bucket* em bytes;
- MConformSize (*Maximum Conform Size*) – número de *tokens* para encher o *bucket*, calculado segundo a Equação 4;

$$MConformSize = CIR * (currentTime - lastTime) + cBucket \quad (4)$$

- MaxPktSize (*Maximum Packet Size*) – tamanho máximo do pacote para que seja considerado conforme;
- MinPolUnit (*Minimum Policed Unit*) – tamanho mínimo usado para decrementar o tamanho do *bucket*;
- Length – tamanho do pacote.

O algoritmo do policiador com *token bucket* funciona da seguinte maneira:

1. Faça currentTime = instante atual;
2. Se o tamanho do pacote for maior que MaxPktSize, descarte-o e saia; senão, continue;
3. Calcule o valor MConformSize. Se este valor for maior que CBS, então MConformSize = CBS;
4. Se MConformSize for menor que zero, então MConforSize = 0;
5. Faça cBucket = MConformSize;
6. Faça lastTime = currentTime;
7. Se o tamanho do pacote for menor que minPolUnit, então faça Length = minPolUnit;
8. Se Length for maior que cBucket, marque o pacote; senão, faça cBucket = cBucket – Length; saia.

5.1.3.2. As Mensagens de Controle

O RSVP-TE utiliza-se de mensagens de controle para sinalização (BRADEN, 1997; AWDUCHE *et al*, 2001). Estas mensagens são geradas pelos roteadores e contém parâmetros e objetos necessários para o correto processamento das sinalizações. O RSVP possui duas mensagens básicas: a PATH e a RESV. Há ainda uma terceira mensagem, adicionada como extensão pelo RSVP-TE, que é a mensagem HELLO. As mensagens são encapsuladas em pacotes de rede. No TARVOS, a estrutura de dados do pacote contém campos específicos que portam as informações de controle.

É interessante observar que o TARVOS batiza as mensagens de controle de modo mais explícito que os indicados nas RFCs, de forma a otimizar seu funcionamento. Uma mensagem PATH que objetiva a formação de um túnel LSP e uma mensagem PATH que indica a ocorrência de um erro, por exemplo, são diferenciadas pelos diferentes tipos de objeto que carregam encapsulados no pacote. Um roteador real identifica estes objetos e processa-os de acordo. No TARVOS, as mensagens são diferenciadas pelos seus nomes: o primeiro caso corresponderia a uma mensagem PATH_LABEL_REQUEST, e o segundo, a uma mensagem PATH_ERR.

As mensagens de controle viajam em um sentido padrão, em relação à posição do gerador de tráfego na topologia, e de acordo com o tipo de mensagem. De maneira geral, as mensagems PATH trafegam no sentido *downstream* (jusante)[1], ou seja, da origem do tráfego para o destino; as mensagens RESV trafegam no sentido inverso, *upstream* (montante), ou do destino do tráfego para sua origem. As exceções notáveis são as mensagens PATH_ERR e RESV_ERR. A Figura 7 ilustra alguns tipos de mensagem e o sentido em que trafegam.

[1] Os termos “montante” e “jusante”, apesar de darem um significado claro para o sentido de tráfego, são estranhos ao jargão de redes de computadores, que prefere os termos no original em inglês “upstream” e “downstream”. Assim será mantido neste documento.

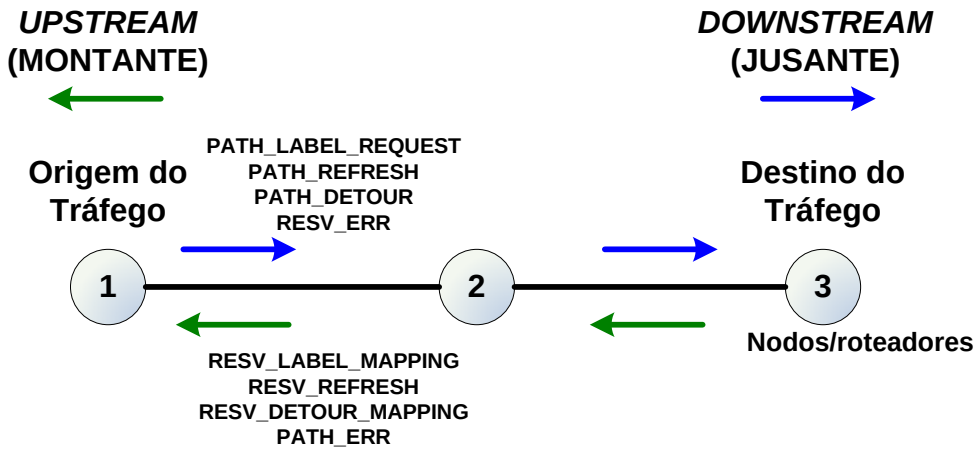

Figura 7: Mensagens de Controle do RSVP-TE (como vistas pelo TARVOS) e seus sentidos de envio.

O TARVOS contempla as seguintes mensagens de controle:

- PATH_LABEL_REQUEST: contrói uma LSP primária e faz pré-reserva de recursos;
- RESV_LABEL_MAPPING: confirma as pré-reservas de recursos de uma LSP primária e faz mapeamento de rótulos;
- PATH_REFRESH: restabelece (*refresh*) os estados do caminho explícito de uma LSP;
- RESV_REFRESH: restabelece (*refresh*) os recursos de uma LSP, confirmando recebimento de uma mensagem PATH_REFRESH;
- HELLO: verifica que o nodo adjacente está acessível;
- HELLO_ACK (HELLO *acknowledge*): confirma que o nodo adjacente recebeu uma mensagem HELLO;
- PATH_DETOUR: monta uma LSP de *backup* e faz pré-reservas de recursos;

- RESV_DETOUR_MAPPING: confirma as pré-reservas de recursos e faz mapeamento de rótulos para uma LSP *backup*, confirmando o recebimento da mensagem PATH_DETOUR;
- PATH_ERR: sinaliza um erro no sentido *upstream*;
- RESV_ERR: sinaliza um erro no sentido *downstream*;
- PATH_LABEL_REQUEST_PREEMPT: mesmas funções da mensagem PATH_LABEL_REQUEST, mas pode usar a preempção, tomando recursos de LSPs de menor prioridade que a LSP corrente;
- RESV_LABEL_MAPPING_PREEMPT: mesmas funções da mensagem RESV_LABEL_MAPPING, mas confirmando também as preempções, se necessárias, removendo LSPs de menor prioridade que a LSP corrente, a fim de tomar-lhe os recursos.

5.1.3.3. Estabelecendo um Túnel LSP

Uma LSP é definida como uma seqüência de roteadores, começando no chamado LER (*Label Edge Router*) de ingresso e terminando no LER de egresso, e para o qual, em todo o percurso, os pacotes seguem comutados por rótulo no mesmo nível da pilha de rótulos (ROSEN; VISWANATHAN; CALLON, 2001). Na concepção do TARVOS, uma LSP compreende vários "pedaços" menores, que são os caminhos entre nodos adjacentes. O caminho completo entre o LER de ingresso e o LER de egresso é chamado, no TARVOS, de *Túnel LSP*. Os túneis são sempre identificados por um número inteiro (chamado LSPid) e de forma única em toda a

topologia de simulação. Os “pedaços” que compõem um túnel LSP (ou seja, os caminhos entre dois nodos adjacentes) recebem o mesmo número identificador de seu túnel.

Para estabelecer um túnel LSP entre dois nodos, o usuário invoca a função *setLSP*, especificando também as restrições que o túnel deve obedecer (os mesmos parâmetros do policiador) e a rota explícita. Ainda, o usuário informa as duas prioridades relacionadas a uma LSP: a prioridades *Setup Priority*, que é a prioridade em relação a tomar recursos; e *Holding Priority*, que é a prioridade em relação a manter os recursos já reservados (AWDUCHE *et al*, 2001). As prioridades são usadas no modo de estabelecimento de uma LSP usando preempção. O simulador então cria uma mensagem tipo PATH_LABEL_REQUEST, encapsula-a em um pacote e envia-a para o nodo de egresso. Se cada salto ou *hop* do caminho é capaz de atender às restrições, então a mensagem PATH atinge o LER de egresso; este LER cria uma mensagem do tipo RESV_LABEL_MAPPING, encapsula-a num pacote e envia-a de volta ao LER de ingresso. Esta mensagem RESV confirma as reservas feitas ao longo do caminho e executa o mapeamento de rótulos, através do preenchimento da tabela LIB (*Label Information Base*) (ANDERSSON *et al*, 2001). As reservas, portanto, são feitas *hop* a *hop.* Quando o túnel LSP está completamente formado e configurado, o policiador de tráfego pode ser ativado de sorte a forçar pacotes a se conformarem com as restrições da LSP.

Se um dos *hops* do caminho explícito indicado para formação do túnel LSP não puder cumprir as restrições, então o nodo envolvido

descartará a mensagem PATH_LABEL_REQUEST. O LER de egresso, então, jamais receberá esta mensagem e não confirmará nenhuma pré-reserva feita em outros *hops*. Estas retornarão à disponibilidade dos enlaces quando o temporizador ligado à LSP indicar estouro do tempo limite de restabelecimento (*refresh*) dos estados (no RSVP-TE, os estados, como as reservas de recursos e túneis configurados, são do tipo *soft state*, ou seja, precisam ser periodicamente restabelecidos, ou sofrerem *refresh*, para que continuem válidos; se não o forem dentro de um tempo limite configurado pelo usuário, os estados são removidos).

5.1.3.4. Estabelecendo um Túnel LSP de Backup

O usuário cria LSPs de *backup*, no método um-para-um (*one-to-one*) (PAN; SWALLOW; ATLAS, 2005), invocando a função *setBackupLSP*. Como parâmetros, fornece o número LSPid da LSP primária (ou original), nodos de origem e destino (chamados de *Merge Points* – MPs) e a rota explícita. O simulador compõe uma mensagem PATH_DETOUR, que percorre a rota explícita pré-reservando recursos, se estiverem disponíveis (os parâmetros das restrições usados são os mesmos da LSP primária indicada). Se os recursos já estiverem reservados para a LSP primária em qualquer parte do caminho, o simulador não faz uma nova reserva; assim, os recursos já reservados são compartilhados entre as LSPs primário e *backup*. Entenda-se que é factível, porque as RFCs não o proíbem, que os caminhos percorridos pela LSP primária e LSPs *backup* podem ser coincidentes em qualquer parte. Obviamente, se uma falha ocorrer um enlace coincidente, então tanto a LSP primária, como o *backup*, sofrerão da interrupção, o que a

princípio derruba a utilidade do *backup*. Porém, a flexibilidade deve existir e está programada no simulador.

Quando a mensagem PATH_DETOUR alcança o MP de destino, isto é um sinal de que a LSP *backup* cumpre, por todo o caminho explícito, os mesmos requisitos da LSP primária; o MP de destino cria uma mensagem RESV_DETOUR_MAPPING e envia pelo caminho reverso para o MP de origem. Esta mensagem tem o efeito de confirmar as pré-reservas e fazer o mapeamento de rótulos para o novo túnel LSP de *backup*.

5.1.3.5. Mantendo os Estados (Soft States) do RSVP-TE

O TARVOS mantém o *soft state* do RSVP-TE com uso das mensagens PATH_REFRESH e RESV_REFRESH. Estas mensagens são geradas periodicamente (o período é configurável) para cada LSP ativa na rede, primária e *backup*. As mensagens HELLO também são geradas periodicamente, por cada nodo da topologia e com destino aos seus nodos adjacentes. Os nodos adjacentes que recebem as mensagens HELLO respondem com uma mensagem HELLO_ACK, indicando, com este ato, que estão ativos e que o enlace entre eles e o emissor da mensagem está operacional. Dois nodos adjacentes, por conseguinte, geram mensagens HELLO entre si, de modo a confirmar a comunicabilidade entre eles nos dois sentidos.

As mensagens HELLO foram concebidas para serem pequenas (20 bytes) e serem geradas em período bem menor que as mensagens de *refresh* das LSPs, de modo que as falhas da rede sejam detectadas mais

rapidamente, mas mantendo baixa sobrecarga na mesma rede (AWDUCHE *et al*, 2001). As mensagens PATH e RESV para *refresh* têm o objetivo de reiniciar os temporizadores de *soft state* da rede. Se houver falha em algum ponto, estas mensagens sofrerão descarte e, assim, haverá estouro de tempo nos temporizadores de *soft state* da rede. O estouro do temporizador é, portanto, um indicativo de uma falha. Quanto menor o período de geração das mensagens PATH e RESV para verificação do estado da rede, mais rápida, conseqüentemente, uma falha será revelada.

As mensagens PATH e RESV, todavia, têm tamanho em torno de 120 bytes (DAVIE *et al*, 2000; PASSAS; SALKINTZIS, 2004). Uma geração muito freqüente destas mensagens para verificação dos estados causaria uma sobrecarga não desprezível na rede, tanto em consumo de banda, quanto em processamento nos roteadores. A mensagem HELLO, de 20 bytes, permite reduzir essa sobrecarga e assim pode ser gerada em períodos curtos. Tipicamente, as mensagens HELLO são geradas, por cada nodo, a cada 5ms. Se uma HELLO_ACK não for recebida em até 3,5 vezes o período de geração (tipicamente, então, em 17,5ms), o nodo que não respondeu (e, por conseguinte, o enlace envolvido) é considerado inalcançável. Todos estes temporizadores são configuráveis pelo usuário no TARVOS.

5.1.3.6. Detecção de Falha na Rede e Recuperação Rápida

Como descrito anteriormente, a detecção de falha na rede é engatilhada por estouros de temporizadores do *soft state* das LSPs ou por erros de recebimento de mensagens HELLO. Quando um nodo é inalcançável devido a um enlace inativo, as LSPs que atravessam este enlace

vão sofrer estouro de seus temporizadores, por não recebimento das mensagens PATH_REFRESH e RESV_REFRESH. Nodos que enviam mensagens HELLO para outros nodos através do enlace falho também não receberão os HELLO_ACKs.

O processo de Recuperação Rápida (PAN, 2005) será ativado tanto por estouro de temporizador de *soft state*, quanto por erro de recebimento de mensagem HELLO. O nodo que detectou a falha tentará encontrar uma LSP de *backup* ou *detour* LSP para cada LSP montada sobre o enlace falho, que trafeguem por uma rota que desvie de tal enlace inativo. O processo, assim, ocorre ao nível do nodo, de modo que nenhuma sinalização é de fato crucial a fim de informar outros nodos ou aplicaçõs de que uma recuperação está em andamento. Para os outros nodos e aplicações, todo o procedimento é transparente. Os efeitos da falha, logicamente, ainda poderão ser sentidos pelas aplicações, visto que alguns pacotes em tráfego pelo enlace poderão ser perdidos no momento da falha.

5.2. Resumo do Capítulo

Descreveu-se, neste Capítulo, a filosofia de construção do simulador de redes TARVOS. Baseia-se ele em três elementos principais: o *kernel*, que contém funções e elementos de redes de filas; o *shell 1*, que contém as entidades básicas de redes de computadores, como enlaces, nodos e geradores de tráfego, e que usa funções do *kernel* para suas implementações; e o *shell 2*, que oferece as funcionalidades do MPLS e

RSVP-TE, e que se utiliza do *shell 1* (e por vezes do *kernel*) para montar suas funções.

Os elementos básicos de redes de filas do *kernel*, que são as *facilities*, os *tokens* e os eventos, foram descritos e sua interrelação esclarecida. As entidades de redes de computadores, a cargo do *shell 1*, foram também pormenorizadas: os nodos, os enlaces, o policiador de tráfego, os pacotes de rede, os geradores de tráfego.

Finalmente, passou-se ao relato das funcionalidades do MPLS e RSVP-TE, a cargo do *shell 2*: estabelecimento de túneis LSP primários e de *backup*, manutenção do *soft state* do RSVP-TE, tipos de mensagens de controle comportadas pelo TARVOS, temporizadores, e os mecanismos de detecção e recuperação rápida de falhas programado no simulador.

6. Preparando uma Simulação

O usuário constrói uma simulação preparando um programa em C, com ao menos a função *main*. Este programa fará uso das funções e estruturas providas pelo TARVOS a fim de modelar a topologia de simulação (nas entidades básicas de redes de computadores, que são os nodos, enlaces, etc.) e também gerenciar uma série de eventos. Os parágrafos seguintes descrevem as quatro etapas primordiais que o programa do usuário deve seguir a fim de se obter uma simulação (Figura 8).

6.1. O Programa do Usuário

6.1.1. Etapa 1: Construindo um Modelo

Primeiro, o usuário prepara um modelo da topologia de rede a simular. Recomenda-se sempre ter à mão um diagrama ou desenho da topologia modelada, identificando com números os nodos, enlaces e geradores de tráfego, bem como outros detalhes desejados. Funções do *shell 1* serão usadas para formar os elementos de redes.

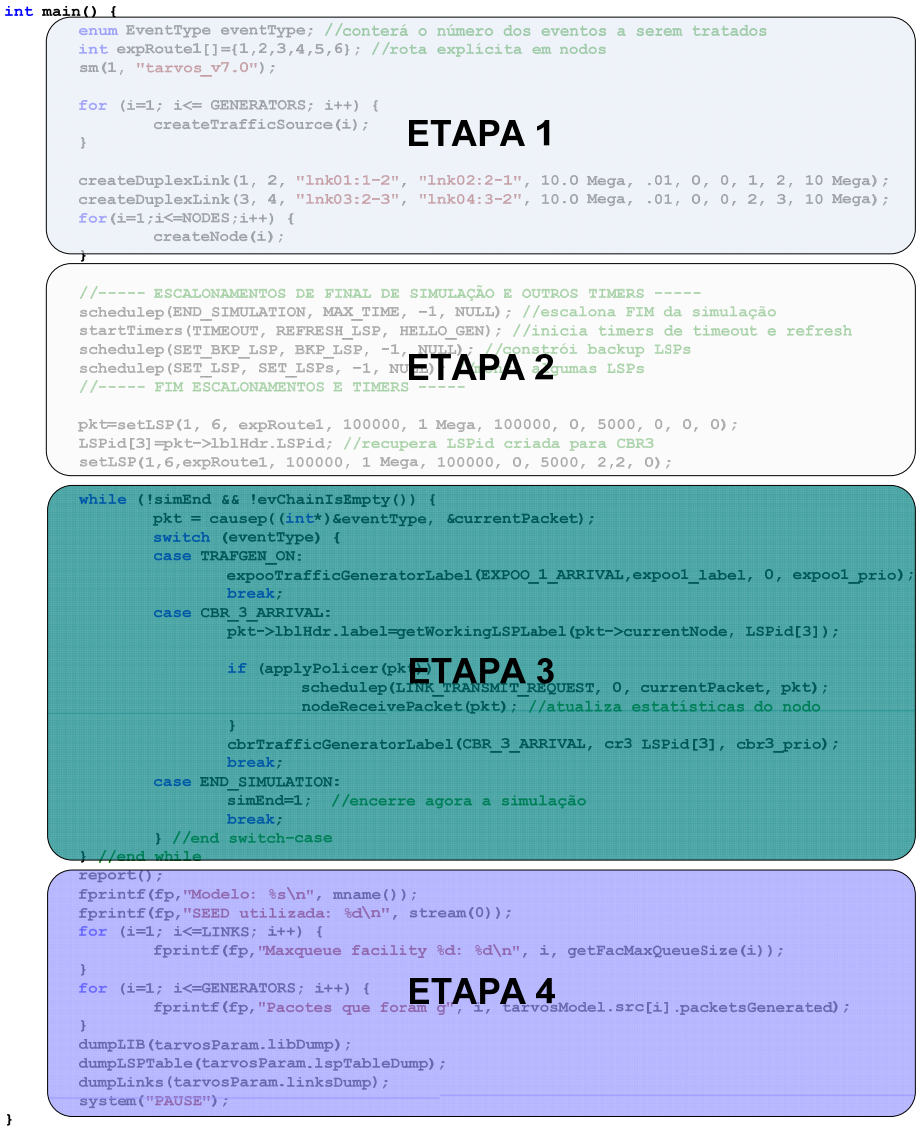

Figura 8: As quatro etapas de um programa de simulação usando o TARVOS.

Assim, para montar o modelo propriamente dito no TARVOS, criam-se, no programa do usuário, os nodos, usando da função *createNode*. Após, criam-se os enlaces que conectarão os nodos, por intermédio da função *createDuplexLink* ou *createSimplexLink*, dependendo do tipo de enlace desejado. Então, ligam-se os geradores de tráfego requeridos nos nodos apropriados, feito pelas funções *createTrafficSource* e as diversas funções de geração de tráfego:

- *expTrafficGenerator:* gerador de tráfego exponencial (ou Poisson);
- *cbrTrafficGenerator*: gerador de tráfego CBR;
- *expooTrafficGenerator*: gerador de tráfego exponencial *On/Off*;
- *expooTrafficGeneratorLabel*: gerador de tráfego exponencial *On/Off* com rótulo inicial para MPLS;
- *cbrTrafficGeneratorLabel*: gerador de tráfego CBR com rótulo inicial para MPLS.

Rotas explícitas e outras estruturas para coletar estatísticas também podem ser definidas nesta etapa.

6.1.2. Etapa 2: Construindo LSPs e Escalonando Eventos Iniciais

Como segundo passo, o usuário constrói os túneis LSP primários e as LSPs de *backup*. A função do *shell 2 setLSP* serve aos túneis LSP primários e a função *setBackupLSP*, às LSPs de *backup*. Ambas funções requerem rotas explícitas para montagem dos túneis.

Nesta etapa, os eventos iniciais e temporizadores da simulação são escalonados, por intermédio da função do *kernel schedulep* (uma contração de *schedule with pointer*). Exemplos de eventos iniciais são: disparo dos temporizadores de *soft state*; começar geração de tráfego; derrubar (tornar inativo) ou levantar (tornar ativo) enlace; reiniciar contadores estatísticos; sinalizar fim da simulação.

6.1.3. Etapa 3: Entrando no Laço de Repetição de Busca e Tratamento de Eventos

Na terceira etapa, o programa do usuário entrará num laço de repetição (*loop*). A cada iteração, retira-se o próximo evento da cadeia de eventos, identifica-se seu tipo e então trata-se o evento de acordo. Volta-se ao início do laço e repete-se todo o ciclo até que o evento de final de simulação (ou a condição de parada) ocorra.

Ordinariamente, esta parte do programa é uma estrutura *switch-case* dentro de um laço *while* (Figura 9). Cada *case* trata um tipo de evento específico chamando funções dos *shells* e, quando apropriado, gerando novos eventos. Observe-se que, se novos eventos não fossem gerados a partir de eventos já existentes, então a cadeia de eventos ficaria vazia, o que equivale a um sistema estático. Quase todos os tipos básicos de eventos geram um outro evento. Por exemplo, a chegada de um pacote ao sistema gera um pedido de transmissão deste ao nodo seguinte, e também gera um novo evento de chegada de pacote ao sistema (em linhas simples, um novo pacote chegando ao sistema "puxa" a chegada de mais um). O pedido de transmissão de um pacote escalona o evento "propagação" do mesmo pacote, após ser transmitido.

6.1.4. Etapa 4: Finalizando a Simulação

Como etapa final, após a sinalização de final da simulação, que encerra o laço de repetição, o programa coleta informações das estruturas de dados e calcula e registra as estatísticas e medidas de desempenho desejadas. Como exemplo, temos atraso, *jitter*, perda de pacotes.

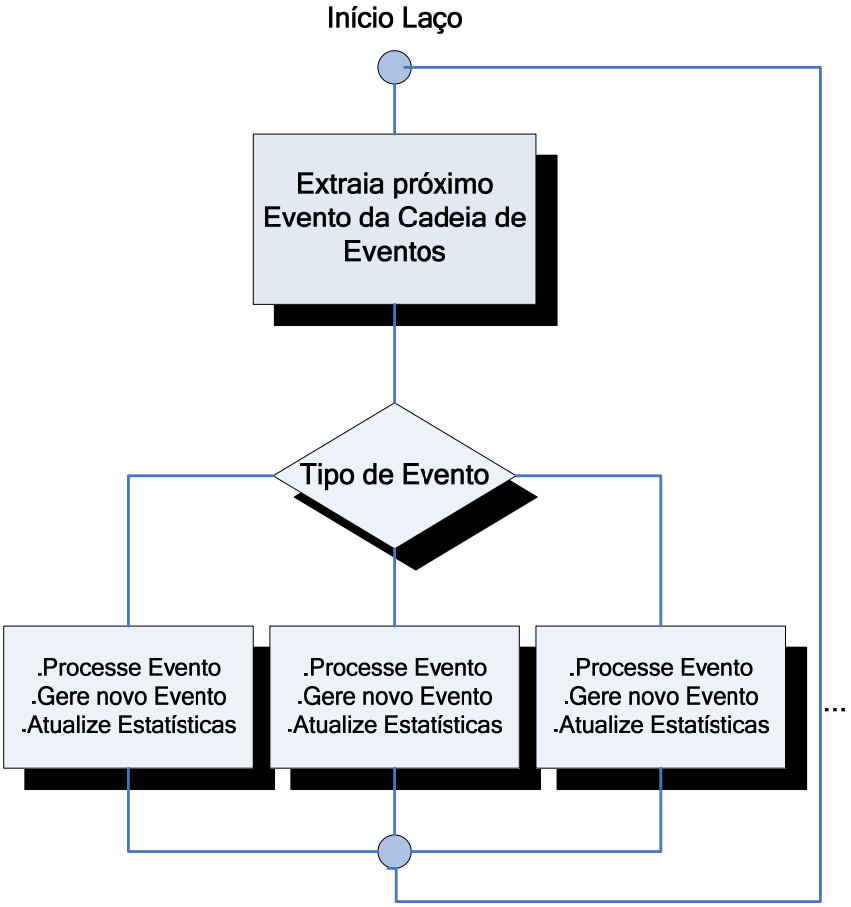

Figura 9: Laço principal de repetição para tratamentos de eventos da simulação.

6.2. O Tratamento de Eventos

Um possível problema de simuladores baseados em eventos, segundo alguma literatura, é sua escalabilidade limitada. Seriam apropriados, pois, para simulações de sistemas pequenos ou médios. O número de eventos que o programa do usuário deveria tratar cresce à medida que uma topologia de simulação torna-se maior.

Por exemplo, seja a topologia ilustrada na Figura 10. Um gerador de tráfego envia pacotes para o nodo 1, que por sua vez encaminha para o nodo 2. Este encaminha para o nodo 3, que é o destino final (indicado pelo

símbolo *sink* ou sorvedouro). Tomando somente a ocorrência de chegada de pacotes a um nodo, teremos, nesta topologia, três eventos distintos: chegada de pacote no nodo 1 (provindo do gerador de tráfego); chegada de pacote no nodo 2 (encaminhado pelo nodo 1); e chegada de pacote no nodo 3 (encaminhado pelo nodo 2). Cada um destes eventos deve ser tratado apropriadamente: imagine-se, então, de acordo com o exposto na Figura 9, que o usuário deve preparar três rotinas separadas para tratar cada evento deste. A chegada de um pacote no nodo 1 deve escalonar o evento de encaminhamento para o nodo 2; a chegada de um pacote no nodo 2 significa escalonar o evento de encaminhamento para o nodo 3, e uma chegada no nodo 3 precisa que estatísticas específicas sejam atualizadas.

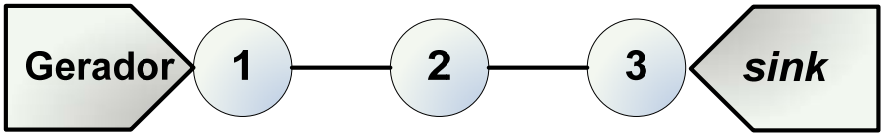

Figura 10: Topologia de rede com três nodos receptores de pacotes.

Se, ao invés de uma topologia com 3 nodos, conceber-se uma topologia com 30 nodos em série, então são 30 eventos distintos do tipo *chegada de pacote a um nodo*. O usuário deveria, consequentemente, preparar 30 rotinas em C para tratamento dos eventos. Se são 300 nodos, serão 300 rotinas somente para este tipo de evento, e por conseguinte. Torna-se fácil perceber como o número de eventos a tratar escalona rapidamente frente ao crescimento de uma topologia.

No TARVOS, fez-se esforço no sentido de limitar o número de tipos de eventos que o programa do usuário deve prever e tratar. Múltiplas funções foram criadas para mimetizar operações de redes e para manter

baixo o número de passos que o usuário necessita processar em cada evento. Desta forma, procurou-se criar "naturezas" de eventos que serão tratados pelo usuário, deixando ao simulador o encargo de verificar o evento específico adequado.

Uma simulação típica tratará basicamente dez tipos de eventos:

1. Chegada de pacote vindo de gerador de tráfego;
2. Pedido de transmissão pelo enlace;
3. Propagar pacote pelo enlace;
4. Chegada de pacote no nodo;
5. Chegada de mensagem de controle;
6. Restaurar (*refresh*) estados das LSPs;
7. Gerar mensagens HELLO;
8. Disparar estouro de temporizadores (*timeout trigger*);
9. Iniciar geradores de tráfego;
10. Finalizar simulação.

As funções e estruturas de dados do TARVOS são responsáveis por quebrar os eventos genéricos listados acima para o nível dos nodos, enlaces ou geradores de tráfego individuais. Por exemplo, o evento *pedido de transmissão pelo enlace* trata genericamente pedidos de transmissão de qualquer pacote, para qualquer enlace. As funções chamadas na rotina de tratamento deste evento, no programa do usuário, detectarão os pacotes e enlaces específicos envolvidos e agirão de acordo. O evento *chegada de pacote no nodo* corresponde à chegada de pacote em qualquer nodo; as

funções do TARVOS verificarão, a partir da informação contida na estrutura de dados do pacote, qual é o nodo em questão e assim disparará os processamentos adequados, que compreendem verificação de rota para encaminhamento, atualização de estatísticas, checagem de temporizadores, entre outras.

Descrevem-se adiante os eventos básicos ou genéricos.

6.2.1. Evento Chegada de Pacote vindo de Gerador de Tráfego

O evento *chegada de pacote vindo de gerador de tráfego* sinaliza a entrada de um novo pacote na topologia ou domínio MPLS. O programa do usuário precisa escalonar um evento do tipo *pedido de transmissão pelo enlace* (chamando uma função de *kernel* do TARVOS) para o pacote que está chegando; escalonar uma nova chegada vinda do mesmo gerador de tráfego (chamando uma função do *shell 1* do TARVOS); e disparar a recepção do pacote pelo nodo corrente (usando de uma função do *shell 2* do TARVOS). Aqui, o policiador de tráfego pode também ser invocado de modo a garantir a conformidade com as restrições de tráfego reservadas.

6.2.2. Evento Pedido de Transmissão pelo Enlace

No evento *pedido de transmissão pelo enlace*, o programa do usuário chama uma função do *shell* do TARVOS para decidir qual o próximo nodo para qual o pacote deve ser enviado. A decisão envolve tanto roteamento explícito, ou comutação por rótulo, ou roteamento estático. Então, chama-se outra função do *shell* para efetivamente transmitir o pacote

(ou colocá-lo na fila de transmissão, caso o enlace esteja ocupado) e escalona-se um evento *propagar pacote pelo enlace*.

6.2.3. Propagar Pacote pelo Enlace

Aqui neste evento, é necessário primeiramente escalonar o fim da transmissão do pacote. Então, ativa-se a propagação propriamente dita. Após, escalona-se um evento *chegada de pacote no nodo*. Todas estas operações são conseguidas através da chamada de duas funções de *shell*.

6.2.4. Chegada de Pacote no Nodo

Este evento invoca a recepção do pacote pelo nodo corrente, ou seja, sinaliza a chegada do pacote no nodo para o qual ele foi transmitido (não necessariamente este é o nodo de destino do pacote, mas pode ser simplemente um nodo qualquer, intermediário). Se o nodo corrente, pois, não for o nodo de destino final do pacote, e o pacote não foi descartado pelo nodo por algum motivo, então escalona-se o evento *pedido de transmissão pelo enlace*.

6.2.5. Chegada de Mensagem de Controle

O processamento do evento *chegada de mensagem de controle* é similar ao processamento do evento *chegada de pacote no nodo*. Invoca-se a função de recepção de pacote pelo nodo e escalona-se, se o pacote não foi descartado, o evento *pedido de transmissão pelo enlace*.

Este é o evento inicial que é gerado quando uma mensagem de controle (como RESV ou PATH) é criada por algum nodo. É a sinalização de que uma mensagem de controle entrou no domínio MPLS.

6.2.6. Restaurar (*refresh*) Estados das LSPs

No evento *restaurar estados das LSPs*, ou fazer o *refresh* de estados das LSPs, o usuário chama a função de *shell* do TARVOS que dispara o *refresh* de todas as LSPs existentes no domínio MPLS. A restauração equivale a emitir mensagens PATH_REFRESH para cada LSP.

É interessante observar que, uma vez que este evento dispara a restauração para todas as LSPs, independente de seus nodos de origem, naturalmente haveria um sincronismo de emissão de mensagens PATH_REFRESH para cada LSP. Este sincronismo pode ser danoso para a rede, pois as mensagens sincronizadas competem por recursos com o tráfego usual. Um mecanismo de atraso aleatório na geração das mensagens PATH_REFRESH é adotado no TARVOS a fim de evitar a sincronização (BRADEN *et al*, 1997).

Este evento também precisa ser re-escalonado para que um novo ciclo de *refresh* ocorra periodicamente.

6.2.7. Gerar Mensagens HELLO

Neste evento, a geração de mensagens de controle HELLO é disparada para todos os nodos da topologia de simulação. Cada nodo gera uma mensagem HELLO para seus nodos adjacentes. Aqui, também se usa algoritmos para evitar a sincronização de emissão das mensagens HELLO

inserindo-se atrasos aleatórios, apesar de não haver restrição ao sincronismo para mensagens HELLO na RFC 3209 (AWDUCHE *et at*, 2001), que é o documento padronizador do RSVP-TE. Na prática, verificou-se que o sincronismo aparentemente pode provocar um comportamento deletério conhecido como *racing condition*[1]. Nesta condição, o resultado de um processo é inesperadamente dependente da sequencia ou temporização de outros eventos. Os eventos, então, tendem a "competir" de forma a ver qual deles influencia o resultado do processo primeiro.

O mesmo evento deve ser re-escalonado a fim de não interromper o ciclo.

6.2.8. Disparar Estouro de Temporizadores (*Timeout Trigger*)

Como o nome sugere, este evento ativa a verificação de estouro de tempo dos seguintes temporizadores:

- *Soft-state* das LSPs, incluindo os túneis e as reservas de recursos;
- Tempo de validade de mensagens de controle PATH. As mensagens PATH ficam armazenadas numa fila de mensagens de controle em cada nodo. Se o tempo limite de validade destas mensagens for atingido, elas são então descartadas e removidas da fila. Uma mensagem RESV serve de resposta a uma mensagem PATH; se a mensagem

[1] Um comentário sobre *racing condition* pode ser lido em http://en.wikipedia.org/wiki/Data_race

PATH não for encontrada, a mensagem RESV também é descartada (o que resulta em não confirmação de recursos ou caminhos, por exemplo);

- Tempo limite de recepção de HELLO_ACKs; a não recepção de um HELLO_ACK indica provavelmente uma falha na rede.

Novamente, o mesmo evento deve ser re-escalonado, usando o período apropriado, de modo que os ciclos de checagem dos temporizadores continuem.

6.2.9. Iniciar Geradores de Tráfego

Geradores de tráfego são os representantes, num simulador, das aplicações ou aplicativos de rede. Eles geram tráfego baseado numa modelagem estocástica do tráfego real da aplicação.

Os geradores de tráfego são iniciados, ou ligados, neste evento. Este evento pode ser escalonado para um intervalo específico, permitindo que, antes que o tráfego de aplicações comece, a topologia possa ser povoada por LSPs primárias e de *backup*. Isso evita que pacotes de aplicação sejam descartados enquanto os túneis LSPs não estão inteiramente operacionais.

6.2.10. Finalizar Simulação

Este evento interrompe o laço de repetição *while*, dentro do qual está a estrutura *switch-case*. Assim, a simulação é encerrada e os resultados podem ser calculados e registrados.

6.3. Resumo do Capítulo

O presente Capítulo aborda a construção de uma simulação usando o TARVOS. A fim de fazê-lo, o usuário prepara um programa em C contendo o modelo da topologia, construindo os túneis LSPs e escalonando os eventos iniciais, montando o laço principal de repetição, onde tratar-se-ão os eventos da simulação, e, finalmente, encerrando a simulação com a coleta e cálculo das medidas de desempenho desejadas. Todos esses procedimentos são obtidos com o uso de funções de *kernel* e de *shell* do TARVOS. Os dez tipos básicos de eventos a serem tratados pelo programa do usuário (Chegada de pacote vindo de gerador de tráfego; Pedido de transmissão pelo enlace; Propagar pacote pelo enlace; Chegada de pacote no nodo; Chegada de mensagem de controle; Restaurar – *refresh* – estados das LSPs; Gerar mensagens HELLO; Disparar estouro de temporizadores – *timeout trigger*; Iniciar geradores de tráfego; Finalizar simulação) são então descritos no Capítulo.

7. Investigação e Validação: Impacto de Falhas de Enlace no Desempenho de uma Aplicação VoIP

O presente estudo avalia o impacto, sobre uma aplicação VoIP, de falhas de enlace, observando a atuação do mecanismo de recuperação rápida de falhas proporcionado pela interoperação dos protocolos MPLS e RSVP-TE. É pertinente, aqui, usar-se algumas páginas para descrever as características do tráfego de rede gerado por uma aplicação VoIP e sua modelagem estatística.

7.1. 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 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. Pode-se entender esses valores intuitivamente da seguinte forma: 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 não é tão bem representado por esta distribuição. Outros modelos foram propostos na literatura com adição de estados a fim de melhorar a representação, mas isso a torna mais complexa.) Durante o período ativo, portanto, os *codecs*[1] (PORTNOI, 2003) estão em ação e cria-se uma geração de dados constante, equivalente a uma fonte de fluxo CBR (*Constant Bit Rate*), com taxa igual à taxa de conversão do *codec* e com pacotes de dados 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

[1] *Codec* é um software ou hardware dedicado que converte um sinal analógico, como a voz, amostrado e quantificado, em uma representação digital, binária, e vice-versa. O nome vem da contração das palavras em inglês *coder-decoder*.

corresponde ao período ativo, e um período inerte. Uma representação esquemática deste tipo de fonte está na Figura 11.

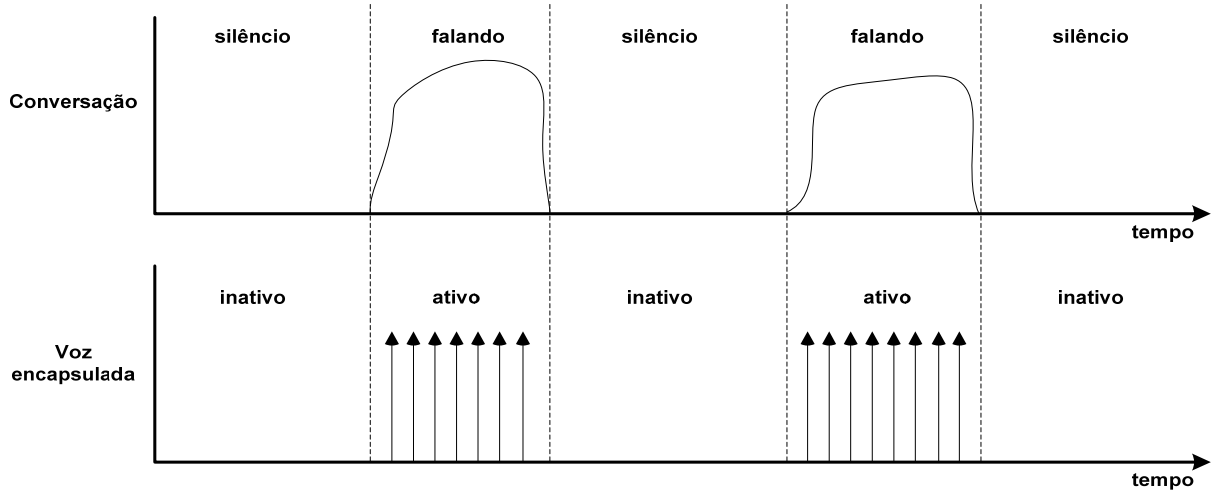

Figura 11: Comportamento de uma fonte de voz típica. Durante os períodos ativos, ou seja, quando o interlocutor está falando, o *codec* gera pacotes de tamanho fixo em intervalos de tempo regulares, indicados pelas setas verticais. Nos períodos de silêncio, não há geração de pacotes.

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; WILLINGER; SHERMAN, 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 12). O parâmetro λ representa a taxa de transição para fora do estado de silêncio (em número de transições

por segundo), e o parâmetro α é a taxa de transição para fora do estado ativo ou *talk spurt* (SCHWARTZ, 1996). Então, a média para o período ativo é 1/α segundos, e o intervalo médio de silêncio é 1/λ segundos.

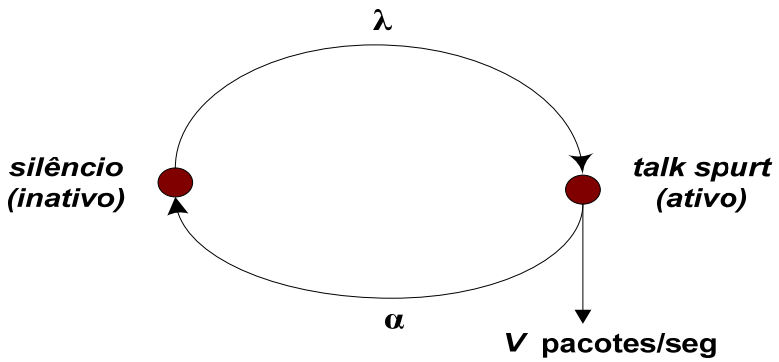

Figura 12: Modelo de dois estados para fonte de voz.

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 de codificação de voz PCM (*Pulse Code Modulation*), esta taxa é de 64 kbps (PORTNOI, 2003).

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 3 traz um resumo destes parâmetros do modelo de voz segundo uma distribuição exponencial *On/Off*.

Tabela 3: 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 usar o modelo mais tradicional.

7.2. O Sistema Estudado: Modelagem e Particularidades

A topologia, vista na Figura 13, foi escolhida para o estudo investigativo. O código em C da simulação está no APÊNDICE A.

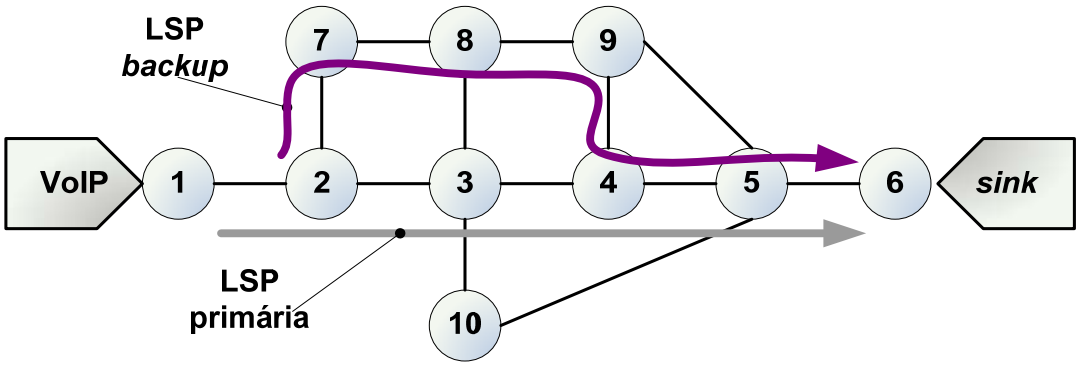

Figura 13: Topologia de teste para o estudo de caso.

A topologia consiste em dez nodos. Um túnel LSP primário, ou protegido, corre através dos nodos 1-2-3-4-5-6. Um túnel LSP de *backup* (*detour LSP*) está montado através dos nodos 2-7-8-9-4-5-6. Dois outros túneis LSP de *backup* têm caminhos 3-8-9-5-6 e 3-10-5-6, respectivamente (Figura 13).

Ligado ao nodo 1, modelando uma aplicação VoIP, está um gerador exponencial *On/Off*. Este gerador envia tráfego para o nodo 6, que é o sorvedouro, ou *sink*, da fonte de tráfego. A fonte gera pacotes de 512 bytes de tamanho a uma taxa de 64kbits por segundo (que é a taxa do *codec* PCM) durante os períodos ON.

Os períodos ON são distribuídos exponencialmente com média 1,2 segundos, e os períodos OFF são distribuídos exponencialmente com média 0,8 segundo. Estes valores das médias são mencionados na literatura como empiricamente satisfatórios para modelar VoIP com uso de um gerador exponencial *On/Off*). Todos os enlaces têm 10Mbps (bps – *bits per second* – bits por segundo) de largura de banda e 10ms de atraso de propagação. Neste estudo, não há nenhum policiador de tráfego em ação. O gerador

exponencial *On/Off* é disparado 5 segundos após o início da simulação (em tempo simulado), a fim de permitir a configuração completa de todas as LSPs.

As mensagens PATH têm um tamanho fixo de 120 bytes (DAVIE *et al*, 2000; PASSAS; SALKINTZIS, 2004) e os estados do RSVP são restaurados a cada 30 segundos (i.e., mensagens PATH_REFRESH de 120 bytes são geradas, para cada LSP, a cada 30 segundos). O estouro do temporizador de estados para o RSVP é de 90 segundos. Mensagens HELLO são geradas em todos os nodos a cada 5ms, e estas mensagens têm 20 bytes de tamanho. O limite de tempo para recebimento de HELLO_ACKs é de 17,5ms (AWDUCHE *et al*, 2001). O simulador faz a checagem de estouro dos temporizadores de estados e mensagens de controle a cada 5ms. Esta simulação é interrompida ao alcançar o tempo simulado de 50 segundos.

A 10,029 segundos do início da simulação, o enlace que conecta os nodos 2 e 3 está escalonado para falhar. O valor do instante de tempo foi escolhido com o propósito de coincidir com um período ON do gerador VoIP. O enlace volta a operar no instante 15 segundos após o início da simulação.

Quando o enlace falha, o tráfego VoIP viajando entre os nodos 1-2-3-4-5-6 é interrompido. O mecanismo de recuperação rápida do domínio MPLS é disparado quando as mensagens HELLO do nodo 2 para o nodo 3 são perdidas e o temporizador de recebimento de ACKs indica estouro de tempo. Localmente, o nodo 2 começa a buscar LSPs primárias estabelecidas no enlace falho (ou seja, que passam pelo enlace defeituoso). Para cada LSP

encontrada, o nodo 2 procura LSPs de *backup* respectivas. Quando uma LSP *backup* é achada, o nodo 2 atualiza sua entrada na LIB, trocando o valor dos campos “enlace de entrada (iIface)” e “rótulo de entrada (iLabel)” para os mesmos valores da LSP primária. Desta forma, os pacotes comutados por rótulo, que seguiriam para o enlace falho, agora trafegarão por um enlace alternativo, desviando da falha. No caso em questão, os pacotes seguirão para o enlace conectando os nodos 2 e 7, e o restante do caminho comutado por rótulo conduzirá os pacotes da LSP corrente pelos nodos 7-8-9-4-5 e 6 (lembrar que a LSP de *backup* está estabelecida entre os nodos 2-7-8-9-4-5-6).

Quando as funcionalidades do enlace falho voltam a operar aos 15 segundos de tempo simulado, as mensagens HELLO emitidas pelo nodo 2 em direção ao nodo 3 percebem a alteração; as mensagens HELLO_ACK voltam a ser recebidas pelo nodo 2. A LSP que originalmente trafegava pelo enlace entre os nodos 2 e 3 não é trazida de volta ao seu túnel inicial, contudo, pois esta capacidade não está programada correntemente no TARVOS.

Duas medidas de desempenho foram coletadas para este estudo de caso: *Atraso de Pacotes de Aplicação*, que é a diferença entre o instante em que um pacote de aplicação VoIP foi enviado pelo gerador de tráfego e o instante em que foi recebido pelo nodo 6, em segundos; e *Jitter de Pacotes de Aplicação*, que é a variação do atraso entre dois pacotes de aplicação VoIP recebidos no nodo de destino – o nodo 6 – em segundos. Em outras

palavras, o atraso e *jitter* dos pacotes exclusivamente gerados pela fonte VoIP foram calculados e registrados no destino (o nodo 6).

7.3. O Atraso

Como se vê na Figura 14, o atraso é computado como sendo 0,052048s ou aproximadamente 52ms antes da falha no enlace. Este valor é a soma do atraso de propagação de todos os enlaces no caminho do pacote (cinco enlaces, no total) com os tempos de transmissão na entrada de cada enlace. Pela Equação 3, o tempo de transmissão nodo-a-nodo (transmissão mais atraso de propagação) de um enlace da topologia de teste é de (512*8)/10 000 000 + 0,01 = 0,0104096. Multiplicado por cinco enlaces entre origem e destino, obtem-se o valor indicado de 0,052048. Notar que, neste caso, não há tráfego competindo com o gerador VoIP, a não ser as mensagens de controle HELLO, PATH e RESV emitidas periodicamente.

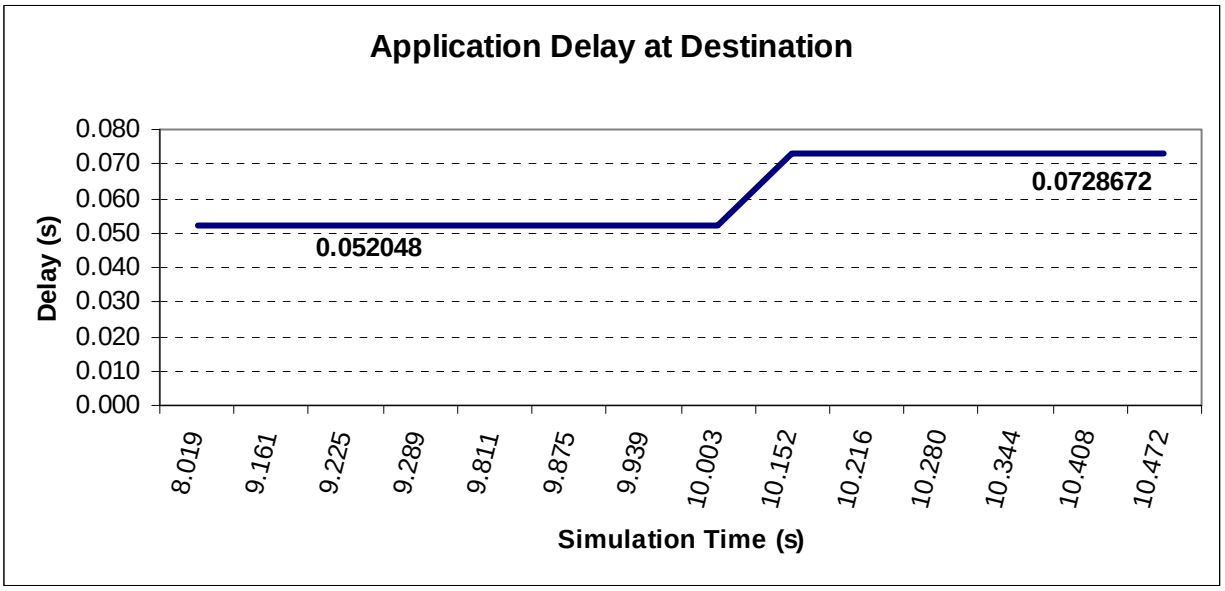

Figura 14: Atraso (*delay*, no eixo y, em segundos) para pacotes de aplicação medido no destino (o eixo x é o tempo simulado ou *simulation time*, em segundos).

Durante os períodos ON, um pacote é recebido pelo nodo destino a cada 0,064s (como demonstra os valores no eixo horizontal da Figura 14). Isto é coerente: a fonte gera pacotes de 512 bytes a uma taxa de 64kbps, o que representa um tempo de intergeração de 0,064s (Equação 5).

$$\text{Tempo de Interger?ção} = \frac{512 * 8\ bits}{64000\ bits/s} = 0{,}064s \quad (5)$$

Neste cenário vislumbrado na Figura 14, o nodo 6 perceberá problemas com a comunicação a partir da aplicação VoIP depois do pacote que é recebido em 10,003s. O próximo pacote é previsto para chegar em 10,067s; porém, um novo pacote somente é recebido em 10,152s, representando um atraso extra de 85ms para este pacote. A partir daí, os tempos de interchegada voltam ao valor esperado de 0,064s durante os períodos ON.

Compreende-se este comportamento conforme segue. Entre os instantes 10,003s e 10,152s, o mecanismo de recuperação rápida entrou em operação localmente no nodo 2, comutando o tráfego de seu caminho original para a rota de *backup*. A rota de *backup*, desde a fonte de tráfego até o destino, trafega pelos nodos 1-2-7-8-9-4-5-6, um caminho mais longo que a rota primária original 1-2-3-4-5-6. Esta rota de *backup* contém mais dois enlaces de 10Mbps e 10ms de atraso de propagação, o que se reflete no novo valor de atraso após a falha mostrado na Figura 14, de 0,0728672s. O valor representa o tempo de transmissão nodo-a-nodo de 0,0104096s (conforme calculado anteriormente), agora multiplicado por sete enlaces. O

simulador também reportou, nesta simulação, que 5 pacotes foram descartados entre os nodos 2 e 3 no momento da falha do enlace, incluindo mensagens de controle e pacotes de aplicação.

7.4. O *Jitter*

Na Figura 15, a medida do *jitter* nos pacotes de aplicação recebidos no nodo 6 mostra basicamente nenhuma variação do atraso, ou *jitter* zero, antes da falha no enlace. Este comportamento é compreensível, visto que a aplicação VoIP é o único tráfego na rede, não competindo com nenhum outro tráfego a não ser a geração periódica de mensagens PATH, RESV e HELLO nos nodos.

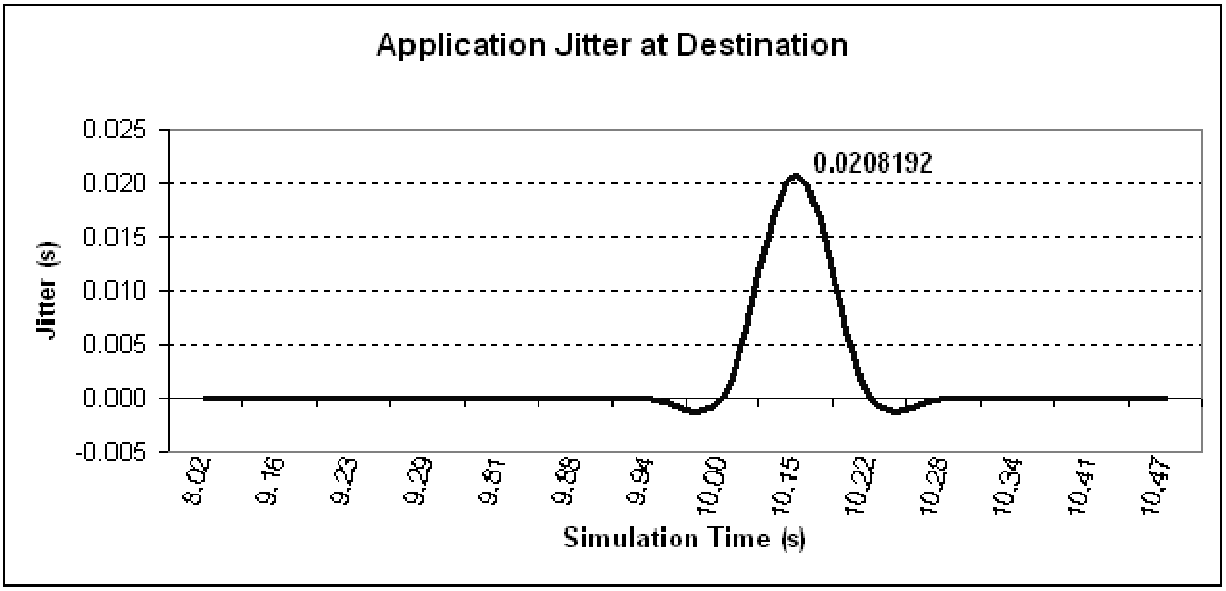

Figura 15: *Jitter* para pacotes de aplicação medido no destino (em segundos).

No exato momento em que o primeiro pacote, após a falha, foi recebido no nodo 6, o *jitter* foi gravado em 20,8ms aproximadamente (Figura 15). Adiante, o *jitter* retorna novamente a zero. O valor 20,8ms, não

coincidentemente, é a diferença entre o atraso antes da falha (52ms) e o atraso após a falha (72,8ms).

7.5. Impacto na Qualidade de Serviço

Conforme visto na Seção 7.1, as recomendações G.144 do ITU-T, para aplicações VoIP (GOODE, 2002), instruem que valores de atraso no destino entre 0 e 150ms são aceitáveis. Para valores de *jitter*, um guia (MIRAS, 2002) sugere que números abaixo de 40ms tornam o *jitter* imperceptível para o usuário. *Jitter* entre 40 e 75ms ainda mantém a aplicação de VoIP com boa qualidade de desempenho, mais sua existência já é notável. Finalmente, valores de *jitter* acima de 75ms seriam inaceitáveis.

No caso estudado, os atrasos medidos estiveram sempre inferiores às recomendações do ITU-T. Obviamente, a topologia usada não exibe tráfego concorrente, o que favorece o ambiente controlado. Sem tráfego concorrente, o atraso mínimo é diretamente dependente das características dos enlaces envolvidos. Quanto maiores suas larguras de banda e menores os atrasos de propagação, então menor o atraso exibido no destino para a aplicação.

Para o *jitter*, as medições mostraram zero na maior parte da simulação, exceto no momento em que ocorre a falha do enlace e o mecanismo de recuperação rápida entra em ação. Neste instante, o *jitter* atingiu a vizinhança de 20,8ms. Segundo (MIRAS, 2002), este valor pode ser considerado indetectável para o usuário de uma aplicação VoIP.

É importante notar, porém, que o valor de pico do *jitter* ocorre apenas uma vez. Após a ação do mecanismo de recuperação rápida de falha, o *jitter* volta velozmente ao seu valor médio de zero. O valor de pico do *jitter* pode ser creditado à variação no atraso provocada pela perda de pacotes, devido à falha no enlace, e também ao acréscimo de atraso causado pelo caminho mais longo seguido pelos pacotes na LSP de *backup*. Tão logo a seqüência de recuperação rápida é completada, o tráfego VoIP recomeça sem mais disrupções.

Note-se que o processo de recuperação rápida acontece quase que instantaneamente, pois ocorre localmente no nodo e depende somente do poder de processamento do roteador envolvido. Nenhuma sinalização da rede é necessária para o mecanismo, desde que logicamente os caminhos de *backup* já estejam montados previamente à falha.

7.6. Resumo do Capítulo

Este Capítulo versa sobre o estudo investigativo feito com uso da ferramenta de simulação TARVOS, protótipo concebido para esta Dissertação. O sistema simulado compõe-se de vários nodos, onde estão montadas uma LSP primária e algumas LSPs de *backup*. Uma falha num dos enlaces por onde corre a LSP primária foi programada, no intuito de analisar o comportamento da mecânica de recuperação rápida do protocolo MPLS, em conjunto com o RSVP-TE, e como ele influi no tráfego, observado no destino, de uma aplicação VoIP.

Descreveu-se, em pormenores, as características de tráfego de uma fonte VoIP e como ela é modelada, probabilisticamente, sob um sistema de simulação. Então, o sistema para estudo foi apresentado, ilustrando sua topologia, seus parâmetros (como quantidade de nodos e perfil dos enlaces). Duas medições foram feitas na simulação: atraso e *jitter*, a partir do nodo de destino e exclusivamente para pacotes de aplicação (desconsiderando, pois, pacotes de sinalização e controle). As medições foram feitas continuamente, começando antes da falha programada e terminando ao final da simulação, de modo que o comportamento das duas variáveis fosse evidenciado antes, durante e após a falha.

A simulação demonstrou que a falha no enlace causou disrupção no tráfego sentido no destino, especialmente visível nos registros de *jitter*. A ação do mecanismo de recuperação rápida ficou entretanto evidente, pois a interrupção no tráfego foi limitada a um momento breve, com alguns pacotes perdidos. Velozmente, o tráfego VoIP retomou sua cadência esperada.

A comparação dos resultados aqui encontrados com o comportamento de uma aplicação VoIP em condições reais implica na montagem e configuração da topologia em laboratório e uso de equipamentos de medição; estes procedimentos fogem ao escopo do presente trabalho.

8. CONCLUSÃO

Nesta Dissertação, proveu-se a descrição da construção de um protótipo de simulador de redes de computadores, denominado TARVOS *Computer Networks Simulator*. Como motivadores para esta construção, havia o interesse inicial de se investigar o impacto de falhas de enlace numa aplicação VoIP funcionando sobre uma rede MPLS com RSVP-TE, e o resultado do estudo com um número de ferramentas de simulação disponíveis. As ferramentas foram averiguadas no tocante a satisfazer uma série de pontos desejáveis, como curva de aprendizado, qualidade da documentação (clareza, abrangência, didatismo, etc.), flexibilidade, disponibilidade livre ou gratuita e inclusão dos protocolos necessários. Como nenhuma satisfizesse todos os pontos, serviram de impulso para o desenvolvimento do protótipo.

Abordou-se as características gerais do protocolo MPLS e do Roteamento Baseado em Restrições, que é funcionalidade cumprida pelo protocolo RSVP-TE. Passou-se então à análise pormenorizada de uma série

de simuladores disponíveis, ressaltando questões que poderiam ser satisfeitas pelo protótipo construído.

Descreve-se, então, a estrutura do TARVOS, seus atributos, as funções que possui e os passos para que um usuário confeccione uma simulação. Durante a descrição, demonstram-se vários dos recursos incorporados no TARVOS. Faz-se, enfim, um estudo investigativo do efeito de falhas de um enlace de rede no desempenho de uma aplicação VoIP, onde também se observa o funcionamento do mecanismo de recuperação rápida do conjunto MPLS/RSVP-TE. Este estudo presta-se ainda a validar o funcionamento do protótipo.

Ressaltam-se, como pontos positivos deste trabalho, que o protótipo TARVOS, procurando atacar quesitos onde os outros simuladores sugeriam melhora, revelou-se uma ferramenta que permite um nível de precisão e detalhamento no tocante a resultados e controle da simulação digno de apreço. Consegue-se regular e vislumbrar a simulação no plano dos eventos principais (graças à orientação a eventos), permitindo gerar estatísticas em tempo de simulação. O uso da linguagem C permite que os códigos resultantes para uma simulação possam ser compactos e também transportáveis para outros sistemas computacionais com pouca ou nenhuma modificação.

A melhorar, têm-se alguns detalhes concernentes ao manuseio de variáveis e estruturas globais que ainda não se revelam simples ou transparentes para o usuário. O número de naturezas de eventos que o

usuário controla, ainda que proporcionem um grau de controle apurado, talvez possa ser otimizado. O único protocolo de transporte simulado, por hora, é o UDP.

Como perspectivas de trabalhos futuros, há a incorporação da simulação do protocolo TCP ao TARVOS; geradores de tráfego auto-similar, que modelam aplicações de vídeo; disponibilização de múltiplas filas para uma *facility* e de mais disciplinas de fila, como WFQ (*Weighted Fair Queueing*), WRR (*Weighted Round Robin*), RR (*Round Robin*), dentre outros; reconstruir as estruturas de dados para permitir múltiplas simulações e a geração automática de intervalos de confiança; incluir protocolos de roteamento, como o OSPF; e, finalmente, produzir uma documentação mais detalhada e abrangente da ferramenta.

Para fins de validação adicionais, sugere-se também a comparação de resultados apresentados pelo simulador com comportamento de topologias similares no universo real.

9. REFERÊNCIAS

ABOUL-MAGT, Osama; JAMOUSSI, Bilel. QoS and service interworking using Constraint-Route Label Distribution Protocol (CR-LDP). **IEEE Communications Magazine**, p. 134-139, maio 2001.

ADAMI, D. *et al.* Signalling protocols in diffserv-aware MPLS networks: design and implementation of RSVP-TE network simulator. In: IEEE Global Telecommunications Conference (GLOBECOM'05). **Anais...** 2005, vol. 2, p. 792-796.

ANDERSSON, L. *et al.* **LDP Specification**. IETF RFC 3036. Jan. 2001.

ATHURALIYA, S. A.; SIRISENA, H. An enhanced token bucket marker for Diffserv networks. In: 12th IEEE International Conference on Networks (ICON 2004). **Anais...** 2004, vol. 2, p. 559-565.

AVICI Systems, Inc. **Traffic Engineering with multiprotocol label switching**. White paper. Avici Systems, 2000. Disponível em: http://www.avici.com/technology/whitepapers/mpls_wp.pdf. Acesso em: Abril 10, 2005.

AWDUCHE, D. *et al.* **Requirements for Traffic Engineering over MPLS.** IETF RFC 2702, Set. 1999.

AWDUCHE, D. *et al.* **RSVP-TE: Extensions to RSVP for LSP Tunnels.** IETF RFC 3209. Dez. 2001.

AWDUCHE, D.; HANNAN, A.; XIAO, X. **Applicability statement for extensions to RSVP for LSP-tunnels.** IETF RFC 3210, Dez. 2001.

BERGER, L. *et al.* **RSVP refresh overhead reduction extensions.** IETF RFC 2961, Abril 2001.

BOERINGER, R. **RSVP-TE patch for MNS/ns-2**. Disponível em: http://www.ideo-labs.com/index.php?structureID=44. Acesso em Set. 20, 2006.

BRADEN, R. *et al.* **Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification**. IETF RFC 2205. Set. 1997.

BRADEN, R.; ZHANG, L. **Resource ReSerVation Protocol (RSVP) version 1 message processing rules.** IETF RFC 2209, Set. 1997.

BRUCE, Davie: DOOLAN, Paul; REKHTER, Paul. **Switching in IP networks:** IP switching, tag switching, and related technologies. San Francisco: Morgan Kaufmann Publishers, 1998, 255 p.

BUCHLI, M. J. C. *et al.* Policing aggregates of voice traffic with the token bucket algorithm. In: IEEE International Conference on Communications (ICC 2002). **Anais...** 2002, vol. 4, p. 2547-2551.

CALLEGARI, C. VITUCCI, F. **RSVP-TE patch for MNS/ns-2**. Disponível em: http://netgroup-serv.iet.unipi.it/rsvp-te_ns/. Acesso em Set. 20, 2006.

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

COLTUN, R. **The OSPF opaque LSA option.** IETF RFC 2370, July 1998.

DAVIE, B. *et al.* **Integrated Services in the Presence of Compressible Flows.** IETF RFC 3006. Nov. 2000.

DAVIE, Bruce; REKHTER, Yakov. **MPLS Technology and applications.** San Francisco: Morgan Kaufmann Publishers, 2000, 287 p.

DER-HWA, Gan *et al.* **A method for MPLS LSP fast-reroute using RSVP detours**. IETF Internet Draft draft-gan-fast-reroute-00.txt. Out. 2001.

GAEIL Ahn. **MNS (MPLS Network Simulator).** [199-?] http://flower.ce.cnu.ac.kr/~fog1/mns/. Este *link* não está mais disponível.

GAEIL Ahn; WOOJIK Chun. Design and implementation of MPLS network simulator (MNS) supporting LDP and CR-LDP. In: 15th International Conference on Information Networking. **Anais...** 2001, p. 694-699.

GAEIL Ahn; WOOJIK Chun. Design and implementation of MPLS network simulator supporting QoS. In: IEEE International Conference on Networks (ICON 2000). **Anais...** 2000, p. 441-446.

GOODE, Bur. Voice over Internet Protocol. **Proceedings of the IEEE**, vol. 90, n. 9, p. 1495-1517, Set. 2002.

HERZOG, S. **Signaled preemption priority policy element.** IETF RFC 2751, Jan. 2000.

HUNG-YING Tyan. **J-Sim Simulator**. Disponível em: http://www.j-sim.org. Last website update: Jan. 28, 2005. Acesso em: Nov. 28, 2006.

JAIN, Raj. **The art of computer systems performance analysis**. Wiley Professional Computing, 1991

JAMOUSSI, Bilel *et al.* **Constraint-based LSP setup using LDP.** IETF RFC 3212, Jan. 2002.

JAVASIM EXTENSIONS. Disponível em: http://www.info.ucl.ac.be/~bqu/jsim. Acesso em: Nov. 28, 2006.

KATZ, D. *et al.* **Traffic engineering (TE) extensions to OSPF version 2.** IETF RFC 3630, Sep. 2003.

MACDOUGALL, M. H. **Simulating computer systems.** The MIT Press, 1987.

MALKIN, G. **RIP version 2.** IETF RFC 2453, Nov. 1998.

MANKIN, A. *et al.* **Resource ReSerVation Protocol (RSVP) version 1 applicability statement:** some guidelines on deployment. IETF RFC 2208, Set. 1997.

MANTAR, Haci. **Discussion of Token Bucket parameters.** Disponivel em: http://qbone.internet2.edu/bb/Bucket.doc. Acesso em: Set. 19, 2006.

MARTINS, Joberto S. B. **CSPF**. Material didático do curso de Qualidade de Serviço em Redes de Computadores do Mestrado em Sistemas e Computação da Universidade Salvador – UNIFACS. Salvador, BA, 2004b.

MARTINS, Joberto S. B. **MPLS basic principles**. Material didático do curso de Qualidade de Serviço em Redes de Computadores do Mestrado em Sistemas e Computação da Universidade Salvador – UNIFACS. Salvador, BA, 2004a.

MCDONALD, Chris. **CNET Network Simulator v.2.0.10**. University of Western Australia. Disponível em: http://www.csse.uwa.edu.au/cnet. Acesso em: Nov. 28, 2006.

MESQUITE SOFTWARE. Disponível em: http://www.mesquite.com. Acesso em Set. 20, 2006.

MIRAS, Dimitrious. (2002) **Network QoS Needs of Advanced Internet Applications: a survey**. Disponível em: http://www.cs.ucl.ac.uk/staff/D.Miras. Acesso em: Dez. 10, 2006.

MOY, J. **OSPF version 2.** IETF RFC 2328, Apr. 1998.

MOY, J. **OSPF version 2.** IETF RFC 2328, April 1998.

MULLAH, Hakim; MOHAMED, Abbou Fouad. Local path protection/restoration in MPLS-based networks. In: 9th Asia-Pacific Conference on Communications (IPCC 2003), 2003. **Anais...** 2003, p. 620-622.

NIYATO, D; DIAMOND, J.; HOSSAIN, E. On optimizing token bucket parameters at the network edge under generalized processor sharing (GPS) scheduling. In: IEEE Global Telecommunications Conference (GLOBECOM 2005). **Anais...** 2005, vol. 2, 5 p.

NORTEL Networks. **MPLS – an introduction to multiprotocol label switching**. White paper. Nortel Networks, 2001. Disponível em: http://www.nortel.com/solutions/providers/enabling_tech/mpls/collateral/55053.25-04-01.pdf. Acesso em: Abril 10, 2005.

NS NETWORK SIMULATOR. Disponível em: http://www.isi.edu/nsnam. Acesso em Set. 20.2006.

OPNET TECHNOLOGIES. Disponível em: http://www.opnet.com. Acesso em Set. 20, 2006.

PAN, P.; SWALLOW, G.; ATLAS, A. **Fast Reroute Extensions to RSVP-TE for LSP Tunnels**. IETF RFC 4090. Maio 2005.

PASSAS, N.; SALKINTZIS, A. A new approach for fast handovers in mobile multimedia networks. In: IEEE 59th Vehicular Technology Conference (VTC Spring 2004), 2004. **Anais...** May 2004, vol. 5, p. 2972-2976.

PETERSSON, Johan M. O. **MPLS Based Recovery Mechanisms**. Dissertação de Mestrado, University of Oslo, Norway, 2005.

PORTNOI, Marcos. **CR-LDP:** aspectos e funcionamento. Disponível em: http://www.geocities.com/locksmithone/academic/academic-files/cr-ldp_aspectos_funcionamento.pdf. Acesso em: fev. 20, 2005.

PORTNOI, Marcos. **Estudo de Características de Fontes de Tráfego para Redes de Computadores Multi-Serviço.** 2003. 114 f. Dissertação (Engenheiro Eletricista) - Curso de Engenharia Elétrica, Universidade Salvador, Salvador, 2003.

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.

PORTNOI, Marcos; MARTINS, Joberto S. B. TARVOS – an event-based simulator for performance analysis, supporting MPLS, RSVP-TE, and fast recovery. In: XIII Brazilian Symposium on Multimedia and the Web – Webmedia 2007. **Anais...** Oct. 20-24, 2007, vol. 1, p. 222-229.

POSTEL, J. **User Datagram Protocol.** IETF RFC 768, Aug. 1980.

PUQI PERRY TANG; TAI, T.-Y. C. Network traffic characterization using token bucket model. In: Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99). **Anais...** Mar. 1999, vol. 1, p. 51-62.

QIAO, Chumning. **MPLS and traffic engineering**. Disponível em: http://www.cse.buffalo.edu/~qiao/cse620/fall04/MPLS-Fall04-NEW.ppt. Acesso em: Jul. 07, 2005.

REKHTER, Y.; LI, T.; HARES, S. **A Border Gateway Protocol 4 (BGP-4)**. IETF RFC 4271, Jan. 2006.

RICE, Liz. **Protection and restoration in MPLS networks**. Disponível em: http://www.dataconnection.com/network/download/whitepapers/mplsprotwp2.pdf. Acesso em: Jul. 12, 2005.

ROSEN, E. *et al.* **MPLS Label Stack Encoding**. IETF RFC 3032, Jan. 2001.

ROSEN, E.; VISWANATHAN, A.; CALLON, R. **Multiprotocol label switching arquitecture**. IETF RFC 3031, Jan 2001.

SCHILDT, Herbert. **C++ The complete reference.** 3rd ed. Berkeley: Osborne McGraw-Hill, 1998, 1008 p.

SCHULZRINNE, H. **RTP**: a transport protocol for real-time applications. IETF RFC 3550, July 2003.

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

SHARMA, V.; HELLSTRAND, F. **Framework for Multi-Protocol Label Switching (MPLS)-based recovery.** IETF RFC 3469, Fev. 2003.

SHENKER, S.; PARTRIDGE, C.; GUERIN, R. **Specification of guaranteed Quality of Service.** IETF RFC 2212, Set. 1997.

SHENKER, S.; WROCLAWSKI, J. **General characterization parameters for Integrated Service network elements.** IETF RFC 2215, Set. 1997.

SKYPE Ltd. Disponível em: http://www.skype.com. Acesso em: Set. 19, 2006.

SWALLOW, George. MPLS advantages for traffic engineering. **IEEE Communications Magazine**, vol. 37, n. 12, p. 54-57, dez. 1999.

TANENBAUM, Andrew S. **Computer networks.** 3rd ed. New Jersey: Prentice Hall, 1996, 814 p.

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.

TSERN-HUEI LEE. Correlated token bucket shapers for multiple traffic classes. In: IEEE 60th Vehicular Technology Conference (VTC 2004). **Anais...** Sep. 2004, vol. 7, p. 4672-4676.

UNIVERSITY OF SOUTHERN CALIFORNIA, Information Sciences Institute. **Internet Protocol**. IETF RFC 791, Sep. 1981.

UNIVERSITY OF SOUTHERN CALIFORNIA, Information Sciences Institute. **Transmission Control Protocol**. IETF RFC 793, Sep. 1981.

VARGA, András. **OMNET++ Discrete Event Simulation System**. Disponível em: http://www.omnetpp.org. Acesso em: Nov. 29, 2006.

VASSEUR, Jean-Phillippe.; ALI, Z.; SIVABALAN, S. **Definition of a Record Route Object (RRO) node-id sub-object.** IETF RFC 4561, Junho 2006.

VASSEUR, Jean-Phillippe; PICKAVET, Mario; DEMEESTER, Piet. **Network recovery:** protection and restoration of optical, SONET-SDH, IP, and MPLS. San Francisco: Morgan Kaufmann Publishers, 2004, 521 p.

WROCLAWSKI, J. **Specification of the controlled-load network element service.** IETF 2211, Set. 1997.

WROCLAWSKI, J. **The use of RSVP with IETF Integrated Services.** IETF 2210, Set. 1997.

XIPENG, Xiao; HANNAN, A.; BAILEY, B.; NI, L. M. Traffic engineering with MPLS in the internet. **IEEE Network**, vol. 14, n. 2, p. 28-33, mar-abril 2000.

YI Lei; CHUNG-HORNG Lung; SRINIVASAN, Anand. A cost-effective protection and restoration mechanism for Ethernet-based networks: an experiment report. In: Workshop on High Performance Switching and Routing (HPSR 2004). **Anais...** 2004, p. 350-354.

YOUTUBE, Inc. Disponível em: http://www.youtube.com. Acesso em: Set. 19, 2006.

APÊNDICE A – Código da Simulação para o Estudo Investigativo

O código a seguir foi utilizado para gerar as estatísticas e análise apresentados no Capítulo 7.

```
/*
* TARVOS Computer Networks Simulator
* Arquivo example-voip.c
*
* Topology for investigation and validation of Tarvos Simulator
* For details on this simulation, see
<http://www.geocities.com/locksmithone>.
*
* Copyright (C) 2004, 2005, 2006, 2007 Marcos Portnoi
*
* This file is part of TARVOS Computer Networks Simulator.
*
* TARVOS Computer Networks Simulator is free software: you can redistribute
it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* TARVOS Computer Networks Simulator is distributed in the hope that it
will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with TARVOS Computer Networks Simulator.  If not, see
<http://www.gnu.org/licenses/>.
*/

#define MAIN_MODULE //define as variáveis e estruturas, e não apenas as
declara como extern
#include "simm_globals.h"
#include "tarvos_globals.h"

int main() {
    enum EventType eventType; //conterá o número dos eventos a serem
tratados
      int currentPacket = 0; //Especifica o pacote atual colhido da cadeia
de eventos; este valor na realidade é o número da token que é utilizada ao
      longo da simulação.

      int i, pktCounter=1, simEnd=0, label, LSPid[10]={0}; //simEnd (flag)
força o final da simulação
      struct Packet *pkt, *aux;
      int expRoute1[]={1,2,3,4,5,6}; //rota explícita em nodos
      int expRoute2[]={2,7,8,9,4,5,6};
      int expRoute3[]={3,8,9,5,6};
      int expRoute4[]={3,10,5,6};

      char mainTraceString[255]; //string que conterá a linha de trace
gerada
      FILE *fp;
```

```
	fp=fopen("screenOutput.txt", "w");

	about(stdout); //imprime informação de copyright na tela e em arquivo
	about(fp);

	simm(1, "tarvos_v0.70 Investigation");

	stream(SEED); //Seeds the random number generator

	system("mkdir stats"); //cria diretório 'stats' para armazenar
arquivos de estatísticas para este modelo (retirar esta linha se isto não
for necessário)

	//Criacao de todas as fontes do sistema
	for (i=1; i<= GENERATORS; i++) {
		createTrafficSource(i);
	}

	createDuplexLink(1, 2, "lnk01:1-2", "lnk02:2-1", 10.0 Mega, .01, 0,
0, 1, 2, 10 Mega);
	createDuplexLink(3, 4, "lnk03:2-3", "lnk04:3-2", 10.0 Mega, .01, 0,
0, 2, 3, 10 Mega);
	createDuplexLink(5, 6, "lnk05:3-4", "lnk06:4-3", 10.0 Mega, .01, 0,
0, 3, 4, 10 Mega);
	createDuplexLink(7, 8, "lnk07:4-5", "lnk08:5-4", 10.0 Mega, .01, 0,
0, 4, 5, 10 Mega);
	createDuplexLink(9, 10, "lnk09:5-6", "lnk10:6-5", 10.0 Mega, .01, 0,
0, 5, 6, 10 Mega);
	createDuplexLink(11, 12, "lnk11:2-7", "lnk12:7-2", 10.0 Mega, .01, 0,
0, 2, 7, 10 Mega);
	createDuplexLink(13, 14, "lnk13:7-8", "lnk14:8-7", 10.0 Mega, .01, 0,
0, 7, 8, 10 Mega);
	createDuplexLink(15, 16, "lnk15:8-9", "lnk16:9-8", 10.0 Mega, .01, 0,
0, 8, 9, 10 Mega);
	createDuplexLink(17, 18, "lnk17:9-4", "lnk18:4-9", 10.0 Mega, .01, 0,
0, 9, 4, 10 Mega);
	createDuplexLink(19, 20, "lnk19:9-5", "lnk20:5-9", 10.0 Mega, .01, 0,
0, 9, 5, 10 Mega);
	createDuplexLink(21, 22, "lnk21:3-10", "lnk22:10-3", 10.0 Mega, .01,
0, 0, 3, 10, 10 Mega);
	createDuplexLink(23, 24, "lnk23:10-5", "lnk24:5-10", 10.0 Mega, .01,
0, 0, 10, 5, 10 Mega);
	createDuplexLink(25, 26, "lnk25:3-8", "lnk26:8-3", 10.0 Mega, .01, 0,
0, 3, 8, 10 Mega);

	//Criação dos nodos
	for(i=1;i<=NODES;i++) {
		createNode(i);
	}

	//----- ESCALONAMENTOS DE FINAL DE SIMULAÇÃO E OUTROS TIMERS -----
	schedulep(END_SIMULATION, MAX_TIME, -1, NULL); //escalona FIM da
simulação
	startTimers(TIMEOUT, REFRESH_LSP, HELLO_GEN); //inicia timers de
timeout e refresh
	schedulep(SET_BKP_LSP, BKP_LSP, -1, NULL); //constrói backup LSPs
	//schedulep(SET_LSP, SET_LSPs, -1, NULL); //monta algumas LSPs com
preempção
	schedulep(TRAFGEN_ON, START_TRAFFIC, -1, NULL); //restarta geração de
tráfego
```

```
	schedulep(BRING_LINK_DOWN, LINK_DOWN, -1, NULL); //escalona queda de
link
	schedulep(BRING_LINK_UP, LINK_UP, -1, NULL); //escalona reabilitação
do link
	//schedulep(RESET, RESET_TIME, -1, NULL); //escalona reset das
estatísticas
	//----- FIM ESCALONAMENTOS E TIMERS -----

	//pkt=setLSP(1, 6, expRoute1, 1 Mega/8, 10000, 1 Mega/8, 0, 5000, 0,
0, 0);
	pkt=setLSP(1, 6, expRoute1, 100000, 1 Mega, 100000, 0, 5000, 0, 0,
0);
	LSPid[3]=pkt->lblHdr.LSPid; //recupera LSPid criada para o gerador

	while (!simEnd && !evChainIsEmpty()) {
		pkt = causep((int*)&eventType, &currentPacket);
		if ((int)simtime()%10==0)
			printf("simtime:  %lf\n", simtime());
		/* cause (ev,tkn), int *ev, int *tkn, cause () remove a entrada
do topo da lista e retorna o numero do evento ev e
		a_tkn = pacote atual. */

		switch (eventType) {
		case TRAFGEN_ON:
			//inicia geração de tráfego
			//Inicializa as fontes (geradores de tráfego) do modelo
			//Cada fonte deve gerar uma chegada inicial, para que
esta chegada, ou evento inicial, dispare novas chegadas
			//
			//Parâmetros:
			//expooTrafficGeneratorLabel:  int ev, int n_src, int
length, int src, int dst, double rate, double ton, double toff, int label
			//cbrTrafficGeneratorLabel:  int ev, int n_src, int
length, int src, int dst, double rate, int label

			expooTrafficGeneratorLabel(EXPOO_1_ARRIVAL, expoo1_nscr,
expoo1_length, expoo1_src, expoo1_dst, expoo1_rate, expoo1_ton,
expoo1_toff, expoo1_label, 0, expoo1_prio);
			break;

		case EXPOO_1_ARRIVAL:
			/* Processamento do tipo pacote saindo da fonte, ou seja
ele indica quando um pacote efetivamente saira da fonte e ira
			para um nodo indicado pelo currentNode */

			//attachExplicitRoute(pkt, &expRoute1); //liga a rota
explícita a este pacote

			/* Escalona o evento (pacote i) para ser processado pela
fila/servidor da facility neste caso o link de saída */
			pkt->lblHdr.label=getWorkingLSPLabel(pkt->currentNode,
LSPid[3]); //coloca o label inicial no pacote

			//Apagar estas 4 linhas e reativar o policer abaixo, caso
o policer seja desejado
			//-------------
			//Escalona o evento (pacote i) para ser processado pela
fila/servidor da facility neste caso o link de saída
			schedulep(LINK_TRANSMIT_REQUEST, 0, currentPacket, pkt);
			//gera um novo pacote para a fonte 4 e escalona na cadeia
de eventos do simm
```

```
                    nodeReceivePacket(pkt); //atualiza estatísticas do nodo
                    //-------------

                    /*if (applyPolicer(pkt)) { //se ==1, pacote está
conforme; escalone transmissão.  se ==0, pacote está não-conforme e foi
descartado
                          //Escalona o evento (pacote i) para ser processado
pela fila/servidor da facility neste caso o link de saída
                          schedulep(LINK_TRANSMIT_REQUEST, 0, currentPacket,
pkt);
                          //gera um novo pacote para a fonte 4 e escalona na
cadeia de eventos do simm
                          nodeReceivePacket(pkt); //atualiza estatísticas do
nodo
                    }*/

                    /* gera um novo pacote para a fonte 1 e escalona na
cadeia de eventos do simm */
                    //int ev, int n_src, int length, int src, int dst, double
rate, double ton, double toff, int label
                    expooTrafficGeneratorLabel(EXPOO_1_ARRIVAL, expoo1_nscr,
expoo1_length, expoo1_src, expoo1_dst, expoo1_rate, expoo1_ton,
expoo1_toff, expoo1_label, 0, expoo1_prio);
                    nodeReceivePacket(pkt); //atualiza estatísticas do nodo
                    break;

              case LINK_TRANSMIT_REQUEST:
                    /* Situacao de envio de pacote por um link a ser
determinado. Tecnicamente colocacao em servico de um
                    pacote em uma facility link*/

                    //define para que link de saída o pacote neste nodo
deverá ser encaminhado. Esta funcao gera o pkt->outgoingLink
                    if (pkt->er.explicitRoute==NULL) { //não há rota
explícita:  use a LIB
                          if (decidePathMpls(pkt)==0) //decide com base no
MPLS; se ==1, rota não existe, portanto não transmita
                                linkBeginTransmitPacket(LINK_PROPAGATE, pkt);
                    } else { //explicitRoute != NULL; então, há rota
explícita.  Use-a.
                          if (decidePathER(pkt)==0) //decide com base na rota
explícita contida no próprio pacote; se ==1, rota não existe, portanto não
transmita
                                linkBeginTransmitPacket(LINK_PROPAGATE, pkt);
                    }
                    /* Encaminha o pacote para o link de saida, ou seja,
escalona-o na cadeia de eventos do simm com o tempo que
                    terminara o servico ou entao o coloca na fila de espera
da facility */
                    break;

              case LINK_PROPAGATE:
                    /* Liberação do serviço de transmissão do pacote.  Este
evento deve necessariamente encadear para o evento
                       linkPropagate relacionado.  Observar que o pacote
ainda não chegou no próximo nodo, mas apenas foi
                       transmitido para o link e agora deve ser propagado.
                    */
                    //Aqui a expressão tarvosModel.lnk[pkt-
>outgoingLink].facility estava sendo resolvida para ZERO no Borland
C++BuilderX.
```

```
                    //O problema estava acontecendo devido à definição das
variáveis globais do tipo struct quando da própria declaração da
                    //struct, no próprio arquivo .h (struct something {...}
some[10];).
                    //Como o arquivo .h era incluído em todos os módulos,
havia múltipla definição das variáveis globais, causando
                    //confusão para o linker.  O problema foi resolvido
separando as declarações das structs, que permaneceram nos
                    //arquivos .h incluídos em todos os módulos, das
definições das variáveis globais do tipo struct, que foram incluídas
                    //somente no módulo principal (através de diretivas de
pré-processamento) e colocadas como extern nos outros módulos
                    linkEndTransmitPacket(tarvosModel.lnk[pkt-
>outgoingLink].facility, currentPacket); //Finaliza a transmissão
                    linkPropagatePacket(ARRIVAL_NODE, pkt);  //introduz o
atraso de propagação (um simples schedulep)

                    //Isto abaixo foi incluso na função linkPropagatePacket
                    //pkt->currentNode = tarvosModel.lnk[pkt-
>outgoingLink].dst; //atualiza o pacote para o nodo em que ele estara apos
ser encaminhado pelo link
                    break;

            case ARRIVAL_NODE:
                    /* Chegada de um pacote em um nodo.  Deve ser testado se
o nodo é o destino do pacote, ou
                       se é um nodo intermediário.  Atualizar também aqui
estatísticas pertinentes.
                    */

                    /*aux é criada aqui para que as estatísticas sejam
impressas no arquivo;
                    *como a função nodeReceivePacket é chamada antes da
impressão, e esta função
                    *descarta o pacote (freePkt(pkt)), a rotina de impressão
não funcionaria pois o pacote
                    *não mais existe.  Após a impressão, aux é eliminada com
free.
                    */
                    //aux=malloc(sizeof *aux);
                    //*aux=*pkt;
                    if (nodeReceivePacket(pkt)==0) { //se retorno ==0, o
pacote deve ser encaminhado para transmissão
                          //Se retorno ==1, significa que o destino do pacote
é o nodo atual.
                          //Se retorno ==2, significa que o pacote foi
descartado pelo nodo (tipicamente, estouro do TTL).
                          //sprintf(mainTraceString, "Simtime:  %f  Nodo:  %d
pktID:  %d  Size:  %d  TTL:  %d  Pacotes que chegaram:  %d    Atraso:
%f\n", simtime(), aux->currentNode, aux->id, aux->length, aux->ttl,
tarvosModel.node[aux->dst].packetsReceived, simtime()-aux->generationTime);
                          //mainTrace(mainTraceString);
                          schedulep(LINK_TRANSMIT_REQUEST, 0, currentPacket,
pkt); //Escalona o evento (pacote i) para ser processado pela fila/servidor
da facility (ou seja, transmitido para o próximo nodo)
                    }
                    //free(aux);
                    break;

            case CTRL_MSG_ARRIVAL:
```

```
                  /* Processamento de mensagens de controle do protocolo
RSVP-TE */
                  //aux=malloc(sizeof *aux);
                  //*aux=*pkt;
                  if (nodeReceivePacket(pkt)==0)
                        schedulep(LINK_TRANSMIT_REQUEST, 0, currentPacket,
pkt);
                  //jitterDelayTrace(aux->currentNode, simtime(),
tarvosModel.node[aux->currentNode].jitter, tarvosModel.node[aux-
>currentNode].delay);
                  //free(aux);
                  break;

            case BRING_LINK_UP:
                  setDuplexLinkUp(3);
                  break;

            case BRING_LINK_DOWN:
                  sprintf(mainTraceString, "Packets in Transit Queue at
this moment simtime(): %f  packets: %d\n", simtime(),
inq(tarvosModel.lnk[3].facility));
                  mainTrace(mainTraceString);
                  setDuplexLinkDown(3);
                  break;

            case END_SIMULATION:
                  simEnd=1;  //encerre agora a simulação
                  break;

            case TIMEOUT:
                  timeoutWatchdog();
                  schedulep(TIMEOUT, tarvosParam.timeoutWatchdog, -1,
NULL);
                  break;

            case REFRESH_LSP:
                  refreshLSP();
                  schedulep(REFRESH_LSP, tarvosParam.LSPrefreshInterval, -
1, NULL);
                  break;

            case HELLO_GEN:
                  generateHello();
                  schedulep(HELLO_GEN, tarvosParam.helloInterval, -1,
NULL);
                  break;

            case RESET:
                  statReset();
                  break;

            case SET_BKP_LSP:
                  setBackupLSP(LSPid[3], 2, 6, expRoute2);
                  setBackupLSP(LSPid[3], 3, 6, expRoute3);
                  setBackupLSP(LSPid[3], 3, 6, expRoute4);
                  break;

            case SET_LSP:
                  setLSP(1,6,expRoute1, 100000, 3 Mega, 100000, 0, 5000,
2,2, 1);
```

```
                    setLSP(2,6,expRoute2, 100000, 2.1 Mega, 100000, 0, 5000,
2,2, 1);
                    break;

              } //end switch-case
        } //end while
        report();
        fprintf(fp,"Modelo: %s\n", mname());
        fprintf(fp,"SEED utilizada: %d\n", stream(0));
        for (i=1; i<=LINKS; i++) {
              fprintf(fp,"Maxqueue facility %d: %d\n", i,
getFacMaxQueueSize(i));
        }
        for (i=1; i<=GENERATORS; i++) {
              fprintf(fp,"Pacotes que foram gerados fonte %d:  %d\n", i,
tarvosModel.src[i].packetsGenerated);
        }
        for (i=1; i<=NODES; i++) {
              fprintf(fp,"Pacotes received no nodo %d:  %d\n", i,
tarvosModel.node[i].packetsReceived);
              fprintf(fp,"..Pacotes forwarded no nodo %d:  %d\n", i,
tarvosModel.node[i].packetsForwarded);
              fprintf(fp,"..Bytes forwarded no nodo %d:  %f\n", i,
tarvosModel.node[i].bytesForwarded);
              fprintf(fp,"..Pacotes perdidos no nodo %d:  %d\n", i,
tarvosModel.node[i].packetsDropped);
              fprintf(fp,"..Bytes recebidos no nodo %d:  %f\n", i,
tarvosModel.node[i].bytesReceived);
              fprintf(fp,"..Throughput no nodo %d:  %f (bytes/seg)\n", i,
tarvosModel.node[i].bytesReceived/simtime());
              fprintf(fp,"..Ultimo delay no nodo %d:  %f (s)\n", i,
tarvosModel.node[i].delay);
              fprintf(fp,"..mean Delay nodo %d:  %f (s)\n", i,
tarvosModel.node[i].meanDelay);
              fprintf(fp,"..Ultimo jitter nodo %d:  %f (s)\n", i,
tarvosModel.node[i].jitter);
              fprintf(fp,"..mean Jitter nodo %d:  %f (s)\n", i,
tarvosModel.node[i].meanJitter);
        }
        for (i=1;i<=LINKS;i++) {
              fprintf(fp,"Pacotes perdidos na facility %d:  %d\n", i,
getFacDropTokenCount(tarvosModel.lnk[i].facility));
        }
        for (i=1; i<=LINKS; i++) {
              fprintf(fp,"Pacotes em Transito link %d: %d\n", i,
getPktInTransitQueueSize(i));
        }
        dumpLIB(tarvosParam.libDump);
        dumpLSPTable(tarvosParam.lspTableDump);
        dumpLinks(tarvosParam.linksDump);
        system("PAUSE");
}
```

APÊNDICE B – Validação da Modelagem do Enlace

A modelagem do enlace no TARVOS, conforme descrita na Seção 5.1.2.2, foi validada usando o simulador NS-2 para gerar estatísticas de atraso na chegada de pacotes. A topologia de teste pode ser vista na Figura 16.

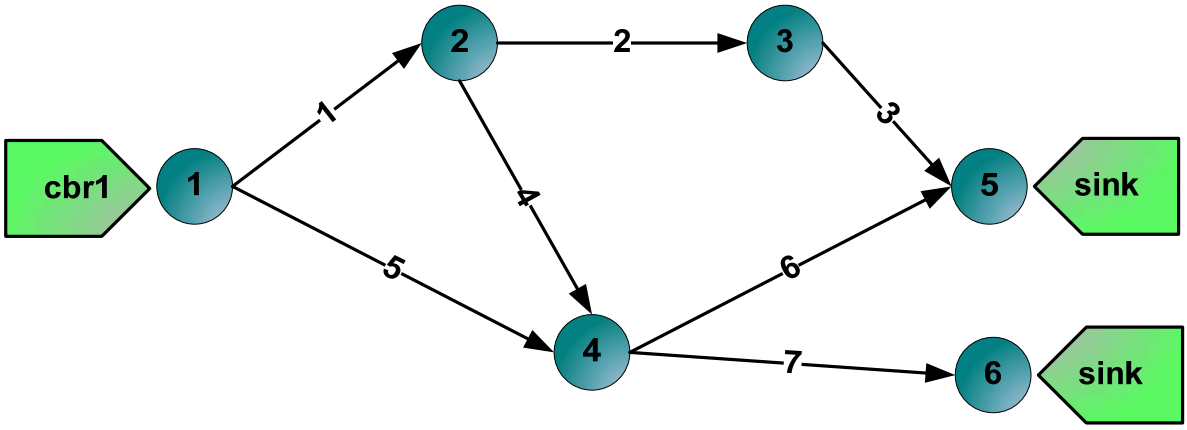

Figura 16: Topologia para validação da modelagem do enlace no TARVOS.

Os enlaces 1, 2 e 3 são de 10 Mbps e com atraso de propagação de 10ms. Os enlaces 4 e 5 são de 1 Mbps, com atraso de propagação de 10ms. Os enlaces 6 e 7 são de 100Mbps com atraso de propagação de 10ms. Um gerador de tráfego CBR é colocado no nodo 1, enviando datagramas de 512 bytes para o nodo 6 a uma taxa de 100kbps. A rota seguida atravessa os nodos 1-2-4-6 (por conseguinte, os enlaces 1, 4 e 7).

Um pacote de 512 bytes demorará, ainda de acordo com a Seção 5.1.2.2, 10,4096ms para chegar ao nodo 2 vindo do nodo 1 (10ms de propagação mais 0,4096ms de transmissão a 10Mbps). Do nodo 2 ao 4, o pacote demorará mais 14,096ms (mais lento, por ser um enlace de 1Mbps). Finalmente, o pacote sofrerá mais um atraso de 10,04096ms entre os nodos

4 e 6, resultando num tempo de atraso total de 34,54656ms. Notar que a taxa de transmissão da fonte CBR não entra nos cálculos aqui, pois a medição só começa quando todo o pacote já está pronto para transmissão pelo enlace.

A simulação foi feita tanto pelo TARVOS, quanto pelo NS-2, coletando-se arquivos de *trace*. Vê-se, a seguir, a semelhança dos resultados apresentados por ambos simuladores.

```
Simtime: 0.075507  Nodo: 6 Pacotes que chegaram: 1  Atraso:  0.034547
Simtime: 0.116467  Nodo: 6 Pacotes que chegaram: 2  Atraso:  0.034547
Simtime: 0.157427  Nodo: 6 Pacotes que chegaram: 3  Atraso:  0.034547
Simtime: 0.198387  Nodo: 6 Pacotes que chegaram: 4  Atraso:  0.034547
Simtime: 0.239347  Nodo: 6 Pacotes que chegaram: 5  Atraso:  0.034547
Simtime: 0.280307  Nodo: 6 Pacotes que chegaram: 6  Atraso:  0.034547
```

Quadro 1: Arquivo de saída do TARVOS.

No Quadro 1, referente ao arquivo de saída do TARVOS, pode-se ver o instante de ocorrência do evento (*simtime*), o nodo em questão, um contador de pacotes que chegaram a este nodo e o cálculo do atraso total do caminho seguindo pelo pacote, que foi de 0,034547s ou aproximadamente 34,547ms.

No Quadro 2, que representa o *trace* do NS-2 (editado para melhor compreensão), vê-se na segunda coluna o instante, em segundos, de ocorrência de chegada de pacote nos nodos (de forma seqüencial). Na 3ª coluna, indica-se o nodo de onde o pacote foi enviado; na 4ª coluna, o nodo onde o pacote está sendo recebido. A 5ª coluna indica o tipo de pacote (CBR) e a coluna seguinte, o tamanho do pacote. As colunas seguintes serão desprezadas aqui.

Assim, observa-se a marcação dos instantes de recepção dos pacotes em cada nodo, e que conferem exatamente com os cálculos feitos anteriormente e com os resultados do TARVOS.

```
r 0.01041 0 1 cbr 512 ------- 0 0.0 5.0 0 0
r 0.024506 1 3 cbr 512 ------- 0 0.0 5.0 0 0
r 0.034547 3 5 cbr 512 ------- 0 0.0 5.0 0 0
r 0.05137 0 1 cbr 512 ------- 0 0.0 5.0 1 1
r 0.065466 1 3 cbr 512 ------- 0 0.0 5.0 1 1
r 0.075507 3 5 cbr 512 ------- 0 0.0 5.0 1 1
r 0.09233 0 1 cbr 512 ------- 0 0.0 5.0 2 2
r 0.106426 1 3 cbr 512 ------- 0 0.0 5.0 2 2
r 0.116467 3 5 cbr 512 ------- 0 0.0 5.0 2 2
r 0.13329 0 1 cbr 512 ------- 0 0.0 5.0 3 3
r 0.147386 1 3 cbr 512 ------- 0 0.0 5.0 3 3
r 0.157427 3 5 cbr 512 ------- 0 0.0 5.0 3 3
r 0.17425 0 1 cbr 512 ------- 0 0.0 5.0 4 4
r 0.188346 1 3 cbr 512 ------- 0 0.0 5.0 4 4
r 0.198387 3 5 cbr 512 ------- 0 0.0 5.0 4 4
```

Quadro 2: Arquivo de saída do NS-2 (editado).

APÊNDICE C – Validação do Gerador Exponencial *On/Off*

Desenvolvido inteiramente por este Autor, a validação deste gerador envolveu o rastreamento de 49.834 pares de valores de tempo (que não serão obviamente aqui listados) coletados numa simulação de teste, com um gerador exponencial *On/Off* configurado em 3,0s de média *On* e 1,0s de média *Off*. Os valores foram capturados diretamente no código do gerador, para que outros tempos inerentes à topologia não fossem computados. O par de valores corresponde a um tempo *On* e um tempo *Off*, calculados pelo algoritmo de geração.

Cada grupo de valores *On* e *Off* representa uma distribuição de probabilidades exponencial. Então, um simples cálculo de média para o grupo de valores *On* e para o grupo de valores *Off* forneceu as médias da distribuição exponencial, resultando, para a simulação em questão, na média 2,9857 para o grupo *On* e 0,9955 para o grupo *Off*. Validado, portanto.

A Figura 17 e Figura 18 trazem os gráficos das distribuições de probabilidade construídas com esta validação.

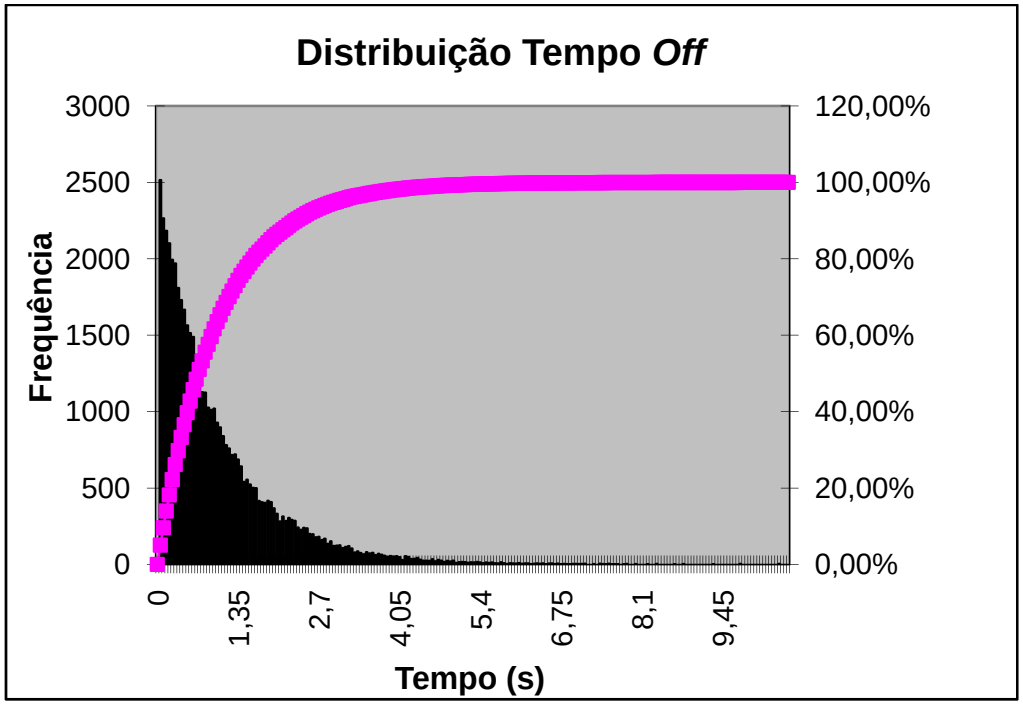

Figura 17: Distribuição do Tempo *Off* coletada para validação do gerador exponencial *On/Off*.

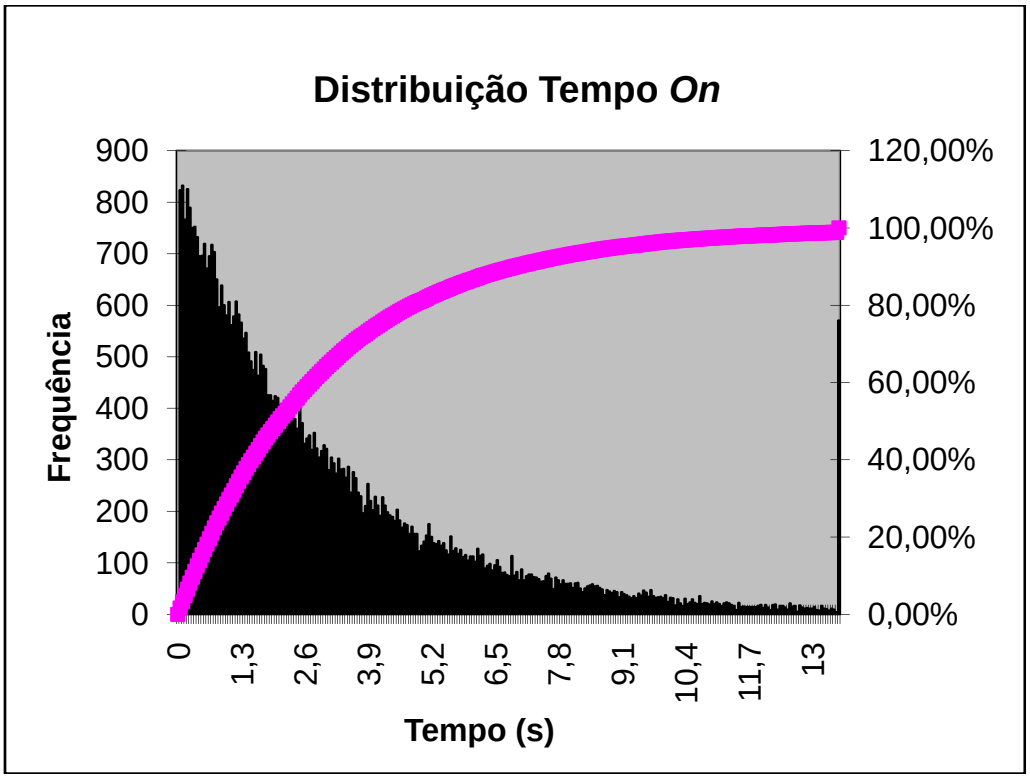

Figura 18: Distribuição do Tempo *On* coletada para validação do gerador exponencial *On/Off*.