UNIFACS
UNIVERSIDADE SALVADOR
LAUREATE INTERNATIONAL UNIVERSITIES

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

RAFAEL FREITAS REALE

ALLOCCT-SHARING - UM MODELO OPORTUNÍSTICO DE ALOCAÇÃO DE BANDA PARA REDES DS-TE

Salvador
2011

RAFAEL FREITAS REALE

ALLOCCT-SHARING - UM MODELO OPORTUNÍSTICO DE ALOCAÇÃO DE BANDA PARA REDES DS-TE

Dissertação apresentada ao Curso de Mestrado Acadêmico em Sistemas e Computação, na Universidade Salvador - UNIFACS, como requisito parcial para obtenção do grau de Mestre.

Orientador: Prof. Dr. Joberto S. B. Martins.

Salvador
2011

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

Reale, Rafael Freitas

Allocct-sharing - um modelo oportunístico de alocação de banda para redes DS-TE / Rafael Freitas Reale. - 2011.
143 f. : il.

Dissertação (Mestrado) - Universidade Salvador – UNIFACS.
Mestrado em Sistemas de Computação, 2011.
Orientador: Prof. Dr. Joberto S. B. Martins.

1. Redes de computadores. 2. Modelo de Alocação de Banda. 3. Qualidade de Serviço (QoS). 4. DiffServ-Aware Traffic Engineering (DS-TE). I. Martins, Joberto S. B., orient. I. II. Título.

CDD: 004.22

TERMO DE APROVAÇÃO

RAFAEL FREITAS REALE

ALLOCCT-SHARING - UM MODELO OPORTUNÍSTICO DE ALOCAÇÃO DE BANDA PARA REDES DS-TE

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre, na Universidade Salvador – UNIFACS, pela seguinte banca examinadora:

Joberto S. B. Martins – Orientador - _____
Doutor em Informática pela Université Pierre et Marie Curie, Paris VI, França
Universidade Salvador - UNIFACS

José Neuman de Souza _____
Doutor em Informática pela Université Pierre et Marie Curie, LISE / CNRS, França
Universidade Federal do Ceará - UFC

José Augusto Suruagy Monteiro _____
Doutor em Ciência da Computação pela University Of California Los Angeles
Universidade Salvador - UNIFACS

Salvador, 14 de outubro de 2011

Dedico este trabalho a toda minha família que sempre deu-me apoio em todas as horas. Ao meu pai, principalmente, que sempre foi um entusiasta deste mestrado, dando-me todo apoio, inclusive o financeiro; à minha esposa, que compreendeu as diversas noites em claro e fins de semana destinados a este trabalho. Também aos meus irmãos pela confiança e torcida.

AGRADECIMENTOS

Em primeiro lugar, agradeço à Deus, por me permitir cursar este mestrado, e por colocar em meu caminho o Prof. Suruagy, que me orientou desde os passos iniciais. Ao Prof. Joberto, ilustre Doutor, por guiar-me até a defesa final deste trabalho. Ao colega Walter por me apoiar nos momentos difíceis. Não posso deixar de mencionar as secretárias do Curso, em especial Aline, pela atenção e compreensão devido ao meu deslocamento constante para o interior. Aos professores e colegas do mestrado que com a convivência somaram no que hoje sou. Ao diretor geral do IFBA – *campus* Valença, Prof. Egberto, e a todos demais colegas, por me compreenderem e me apoiarem nos momentos mais difíceis do mestrado.

RESUMO

Os modelos de alocação de banda são de grande valia para uma utilização eficiente e customizada dos recursos de rede. No contexto das redes DS-TE (*DiffServ-Aware Traffic Engineering*) são propostos modelos que têm como premissa básica a manutenção das prioridades das Classes de Tráfego (CTs) e o respeito das restrições de banda (BC) para cada uma delas. Esta dissertação propõe um novo modelo de alocação de banda, integrado ao CSPF (*Constrained Shortest Path First*), no qual as CTs superiores possam vir a utilizar o tráfego não utilizado de CTs hierarquicamente inferiores e vice-versa. No modelo AllocCT-Sharing preserva-se a garantia mínima prevista nos acordos de serviço (*Service Level Agreement - SLAs*) com uma melhoria da qualidade no atendimento da mesma em diferentes cenários de distribuição de tráfego devido ao maior compartilhamento entre as CT.

Palavras-chave: Modelo de Alocação de Banda. Qualidade de Serviço (QoS). DiffServ-Aware Traffic Engineering (DS-TE).

ABSTRACT

Bandwidth allocation models are important tools towards an efficient allocation of resources in networks. In DS-TE (DiffServ-Aware Traffic Engineering) networks context there are allocation models which fundamentally respect constraints and preserve priorities among defined Traffic Classes (CTs). This dissertation proposes a new bandwidth allocation model called "AllocCT-Sharing" which is integrated with CSPF (Constrained Shortest Path First) routing protocol. In AllocCT-sharing, CTs hierarchically superior may, opportunistically, make use of bandwidth not used by lower priorities CTs and applications. AllocCT-sharing model preserves all service level agreements (SLAs) involved and, beyond that, attempts to improve the overall usability of link resources available in specific traffic distribution scenarios.

Keywords: Allocation Bandwidth Model. Quality of Service (QoS). DiffServ-Aware Traffic Engineering (DS-TE).

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVEATURAS E SIGLAS

AF	*Assured Forwarding*
BCs	*Bandwidth Constraints*
BA	*Behavior Aggregates*
CR-LDP	*Constraint-based Routing Label Distribution Protocol*
CSPF	*Constrained Shortest Path First*
CT	Classes de Tráfego
DSCP	*Differentiated Service Code Point*
DS-TE	*DiffServ-aware MPLS Traffic Engineering*
EF	*Expedited Forwarding*
FEC	*Forward Equivalence Class*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*
IS-IS	*Intermediate System-to-Intermediate System*
ITU-T	*International Telecommunication Union - Telecommunication Standardization Sector*
LDP	*Label Distribution Protocol*
LER	*Label Edge Router*
LIB	*Label Information Database*
LSP	*Label Switched Path*
LSR	*Label Switching Router*
MAM	*Maximum Allocation Model*
MPLS	*Multiprotocol Label Switching*
MPLS-TE	*Multiprotocol Label Switching Traffic Engineering*
MTBF	*Mean Time Between Failures*
MTTR	*Mean Time To Repair*
OSPF	*Open Shortest Path First*
PHB	*Per-Hop Behavior*
QoS	*Quality of Service*
RDM	*Russian Dolls Model*
RFC	*Request for Comments*
RSVP	*Resource Reservation Protocol*
RSVP-TE	*Resource Reservation Protocol Traffic Engineering*
SLA	*Service Level Agreement*

TCP	*Transmission Control Protocol*
TED	*Traffic Engineering Database*
VPN	*Virtual Private Network*

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

A Internet tem alcançado um papel primordial nas comunicações mundiais, suportando uma gama bastante variada de serviços e aplicações. Os princípios da Internet estabeleceram que o núcleo da rede deveria ser simples com inteligência nas extremidades (Princípio Fim-a-Fim). O princípio fim-a-fim possibilitou o desenvolvimento de diversas novas aplicações já que não havia necessidade de intervenções e conhecimento do núcleo da rede. Dentre estas aplicações há uma crescente demanda por desenvolvimento de aplicações de voz, vídeo e entretenimento, como os jogos *online*. Contudo estas novas aplicações esbarram em outra implicação do princípio fim-a-fim, a Internet opera normalmente num estilo "melhor esforço" (*Best Effort*), do qual não existe garantia para os parâmetros de operação da rede. O estilo de operação do melhor esforço (serviço) viabiliza um núcleo da rede simples devido a não trabalhar, por exemplo, com alocação de recursos. No estilo melhor esforço, os pacotes são unidade de dados independentes que podem seguir caminhos e filas diferentes sofrendo, eventualmente, alteração na ordem de chegada, perdas, atrasos e descartes (KUROSE; ROSS, 2000; TANENBAUM, 1998; MOREIRA *et al*, 2009).

As aplicações em tempo real (*real time*) ou interativas, (como aquelas de áudio, vídeo e jogos), precisam, normalmente, de certa garantia de banda, atraso, variação do atraso (*jitter*) e perda de pacotes para operar de uma forma satisfatória para o usuário. Uma grande perda de pacotes, como em uma aplicação de voz, pode haver cortes na comunicação, e um grande atraso pode gerar um desconforto para os interlocutores (KUROSE; ROSS, 2000).

Com intuito de definir uma configuração de operação para a rede que considera as necessidades em termos de banda, atraso e outras métricas, definiu-se o conceito de qualidade de serviço.

A qualidade de serviço em redes IP pode ser entendida como sendo a técnica utilizada para definir o desempenho de uma rede relativa às necessidades das aplicações, como também o conjunto de tecnologias que possibilita às redes de computadores oferecerem garantias de desempenho (MARTINS *et al*, 2003).

Para prover qualidade de serviço (QoS) é necessária a implantação de mecanismos auxiliares à arquitetura TCP/IP, pois esta, em sua proposta inicial, não suporta garantias de desempenho. As soluções atuais existentes e mais frequentemente utilizadas para implantação de um suporte de QoS em redes IP são: superdimensionamento, serviços integrados (IntServ), serviços diferenciados (DiffServ) e MPLS/DS-TE.

O superdimensionamento é a solução na qual se disponibilizam recursos muito além do necessário, como banda, espaço em buffers e capacidade nos roteadores a fim de evitar possíveis disputas. O superdimensionamento é uma forma intuitiva e muito utilizada de obter QoS; porém deve-se levar em conta sua viabilidade financeira (TANENBAUM, 2003).

A solução IntServ (*Integrated Services*) baseia-se no esquema reserva de recursos antes do estabelecimento da comunicação para os fluxos de tráfego. O IntServ é recomendado para pequenas redes ou redes de pequeno porte, pois é preciso sinalizar, processar e manter os estados de cada fluxo que atravessa a rede o que, por sua vez, gera problemas de escalabilidade para redes maiores como a Internet (MONTEIRO; SAMPAIO; FIGUEREDO, 2002; ARMITAGE, 2000).

Os serviços diferenciados comumente chamados de DiffServ (*Differentiated Services*) buscam oferecer QoS baseado na agregação de fluxos em classes de serviço e numa configuração adequada de filas e mecanismos de escalonamento de pacotes nos roteadores. A agregação de fluxos em classes de serviços pré-determinadas oferece escalabilidade para redes de grande porte de maneira geral (MONTEIRO; SAMPAIO; FIGUEREDO, 2002; ARMITAGE, 2000).

Além e complementarmente às soluções básicas indicadas, pode-se fazer uso das técnicas de Engenharia de Tráfego para adequar os recursos de uma rede às suas aplicações. As técnicas de Engenharia de Tráfego possibilitam um mapeamento baseado em critérios de operação da rede dos fluxos de tráfego conforme seus requisitos de QoS com a obtenção de vantagens, tais como a redução dos pontos de congestionamento e uma utilização mais eficiente da rede (AWDUCHE, 1999).

A solução DS-TE (*DiffServ Aware MPLS Traffic Engineering*), foi mais recentemente proposta como uma alternativa de organização de fluxos de tráfego que pode ser utilizada juntamente com soluções tecnológicas para o QoS como DiffServ e o MPLS. A solução DS-TE, obviamente, também pode ser utilizada no escopo das aplicações de engenharia de tráfego (LE FAUCHER; LAI, 2003).

Na solução DS-TE é possível mapear os fluxos, com base em seus requisitos de QoS, em classes de tráfego (CT) que podem ser utilizadas numa estratégia de QoS. Uma das recomendações das redes DS-TE é a adoção de um modelo de alocação de banda para definir as regras que serão utilizadas para a reserva de banda por CT nos enlaces e um algoritmo de escolha do melhor caminho ciente dos requisitos e configuração das CTs. Este é o ponto focal do trabalho desenvolvido e a sua motivação. Nas redes DS-TE, o desempenho da rede como um todo está diretamente associada à escolha correta do modelo de alocação de banda e do algoritmo de seleção de caminhos adequado aos fluxos nela existentes. Os modelos de

alocação de banda e algoritmo de seleção de caminhos necessitam de uma atenção especial nas redes DS-TE, pois um dos principais fatores de degradação da qualidade de serviço é a disputa por banda entre as aplicações em rede.

Os modelos de alocação de banda, por serem os responsáveis pela alocação das bandas na rede, são peças-chave para prover a qualidade de serviço necessária para atender a demanda das aplicações. Existem diversos modelos de alocação de banda e, dentre estes, os dois modelos mais conhecidos são o modelo *Maximum Allocation Model* (MAM - RFC 4125) e o modelo *Russian Doll Model* (RDM - RFC 4127) (LE FAUCHER; LAI, 2005; LE FAUCHER 2005a; PINTO NETO, 2008).

O modelo de alocação de banda MAM define larguras máximas de tráfego para cada uma das classes com total isolamento entre elas, ou seja, uma classe de tráfego não pode usar a banda reservada para uma outra classe. Apesar da vantagem do isolamento entre classes de tráfego, o modelo implica em baixa eficiência na utilização dos enlaces.

O modelo RDM tem como objetivo o compartilhamento de banda não utilizada de classes de tráfego de maior prioridade para classes de tráfego de menor prioridade seguindo o modelo básico de prioridades das aplicações em rede. Devido ao compartilhamento entre classes é necessário um mecanismo que garanta às CTs de maior prioridade a devolução de suas bandas compartilhadas com as CTs de menor prioridade (Preempção) para garantir a qualidade de serviço destas. No modelo RDM, não é previsto o compartilhamento de banda não utilizada de CTs de menor prioridade para CTs de maior prioridade.

Com o uso crescente de aplicações de maior prioridade como áudio, vídeo e jogos em tempo real é desejado, em cenários onde o enlace não esteja saturado, que estas aplicações possam utilizar a banda disponível de CTs de menor prioridade para iniciar novas aplicações ou a fim de melhorar a qualidade de seus serviços. Diante deste novo cenário, observa-se a necessidade de um novo modelo de alocação de banda que permita um maior compartilhamento entre as classes de tráfego de forma tal que em situações de disponibilidade de recursos estes possam vir a ser utilizados também por CTs de maior prioridade, oportunisticamente, e em momentos de conflitos, e apenas nestes, sejam aplicadas medidas restritivas de forma que a QoS seja preservada conforme a configuração previamente estipulada. Assim sendo, a motivação principal é a busca por uma melhor utilização da rede como um todo.

Nesta dissertação, são abordados os dois principais pontos de interesse em redes DS-TE: a escolha de caminho e a alocação de banda entre diversas classes de tráfego, tendo como motivação a melhor utilização do recurso “banda” numa rede.

1.2 CONTRIBUIÇÕES

Este trabalho possui como foco os modelos de alocação de banda devido à direta associação dos mesmos ao desempenho de uma rede DS-TE como um todo. Os modelos de alocação de banda são peça chave, nas redes DS-TE, para prover a qualidade de serviço necessária para atender a demanda das aplicações.

Neste sentido, a principal contribuição deste trabalho é uma proposta de algoritmo de alocação de banda denominado AllocCT-Sharing. O AllocCT-Sharing permite que Classes de Tráfego superiores possam vir a utilizar a banda não utilizada de CTs hierarquicamente inferiores e vice-versa.

Além disto, esta dissertação propõe a integração do algoritmo de cálculo do menor caminho baseado em restrições CSPF (*Constrained Shortest Path First*) ao modelo de alocação de banda AllocCT-Sharing denominando-o "*Loan CSPF*".

A fim de avaliar o comportamento dos algoritmos propostos (AllocCT-Sharing e "*Loan CSPF*") implementou-se em linguagem C uma extensão ao simulador de propósito específico BAMSim (PINTO NETO, 2008).

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

O presente trabalho tem a seguinte organização: (i) o capítulo 2 apresenta uma revisão bibliográfica sobre os conceitos básicos de Qualidade de Serviço (QoS) em Redes Multiserviço e Engenharia de Tráfego (TE - *Traffic Engineering*); (ii) o capítulo 3 apresenta uma revisão bibliográfica sobre os objetivos, a arquitetura e os mecanismos de funcionamento das redes DS-TE, bem como as evoluções das redes MPLS no sentido das redes MPLS-TE; (iii) o capítulo 4 apresenta a proposta dos algoritmos AllocCT-Sharing e "*Loan CSPF*"; (iv) o capítulo 5 apresenta uma prova de conceitos através de simulações no BAMSim e avaliações desta; (v) o capítulo 6 apresenta as considerações finais sobre o trabalho e propostas de trabalhos futuros.

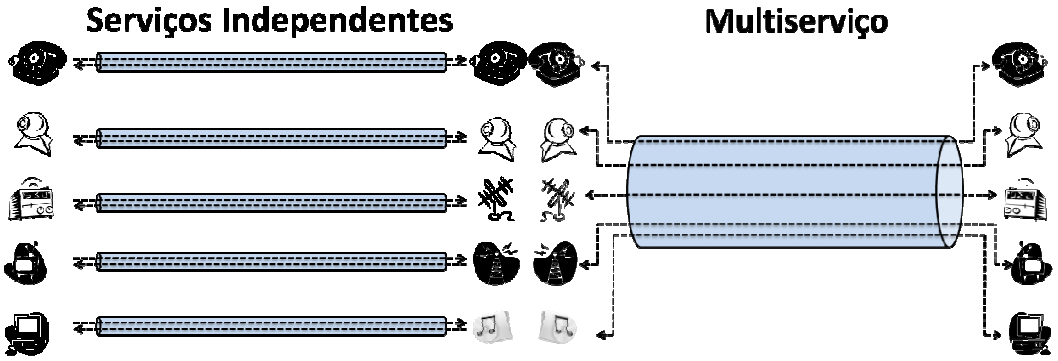
2 QUALIDADE DE SERVIÇO EM REDES MULTISERVIÇO

As redes multiserviços surgem no cenário de crescente demanda por recursos de rede e necessidades diferentes de perfis, computadores e aplicações. As redes multiserviços são uma alternativa para integrar, em uma única infraestrutura de rede, vários serviços e tecnologias de forma mais eficiente.

A utilização de redes multiserviço possibilita a operação de uma única rede, ao invés de redes separadas para cada tipo de serviço, implicando em uma otimização da infraestrutura de *backbone* que, no caso do IP, é de grande interesse hoje (

figura 1).

Figura 1 – Serviços Independentes versus Multiserviço



A integração satisfatória destes diversos serviços em uma rede multiserviço IP implica em considerar que estes podem possuir requisitos distintos de qualidade de QoS (Qualidade de Serviço). Estes requisitos distintos devem ser atendidos por esta infraestrutura de rede única, arbitrando inclusive a concorrência por recursos.

2.1 QUALIDADE DE SERVIÇO

Encontramos na literatura dois conceitos de Qualidade de Serviço com o fim de garantir a satisfação do usuário, bem como o bom funcionamento das diversas aplicações. O primeiro está associado à experiência do usuário em relação à qualidade de uma determinada aplicação; o segundo relaciona-se ao conjunto de parâmetros a serem garantidos às aplicações pela infraestrutura de rede.

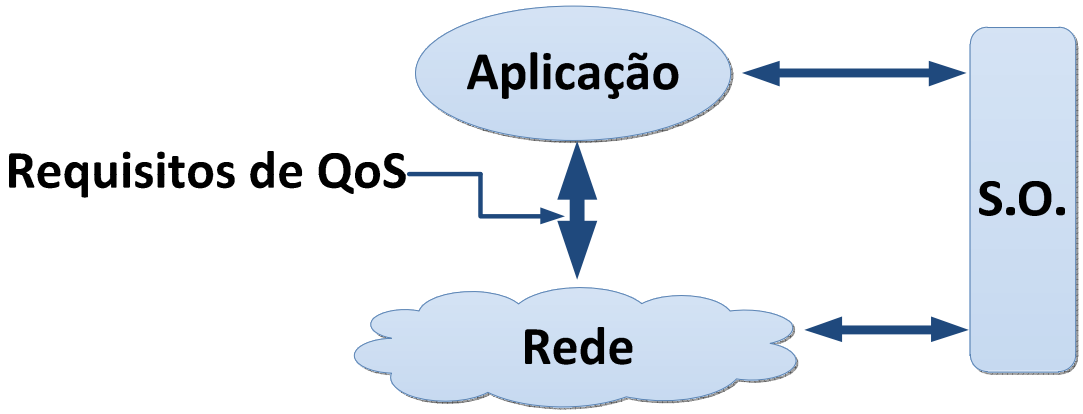
O ITU-T (*International Telecommunication Union - Telecommunication Standardization Sector*), em sua recomendação I.350, define o conceito de qualidade de serviço como o efeito coletivo de desempenho, que determina o grau de satisfação do usuário deste serviço específico. O conceito de qualidade de serviços como definido pelo ITU-T está associado à

percepção do usuário sobre a qualidade de serviço das aplicações e vem sendo denominado de QoE (*Quality of Experience*).

Como este trabalho tem como foco o conjunto de tecnologias que possibilitam às redes de computadores oferecerem garantias de desempenho, chamaremos a Qualidade de Serviço como sendo a técnica utilizada para definir o desempenho de uma rede relativa às necessidades das aplicações, como também o conjunto de tecnologias que possibilita às redes de computadores oferecerem garantias de desempenho (MARTINS *et al*, 2003).

Segundo Martins (1999), a QoS é um requisito da(s) aplicação(ões) (figura 2) para a qual se exigem determinados parâmetros de desempenho (banda, atraso, *jitter* e perda de pacotes) dentro de limites bem definidos para um desempenho satisfatório da rede.

Figura 2 – Requisitos de QoS



Fonte: Martins *et al (* 2003).

2.1.1 Requisitos de QoS

Os requisitos de QoS são definidos comumente a partir das seguintes métricas de desempenho da rede:

- Banda (*Bandwidth*): É a capacidade dos canais de comunicação disponível para a aplicação em todo caminho do seu fluxo de dados. Na tabela 1 é exemplificada a banda típica para algumas aplicações em rede.
- Vazão (*Throughput*): É a banda efetiva recebida da rede. Podemos avaliá-la quanto à vazão mínima, média e máxima e suas rajadas quanto a volume e duração.

Tabela 1 - Banda Típica para Algumas Aplicações em Rede

Aplicação	Faixa de Banda
Aplicações Transacionais	1 Kbps a 50 Kbps
Quadro Branco (*Whiteboard*)	10 Kbps a 100 Kbps
Voz	10 Kbps a 120 Kbps
Aplicações Web (WWW)	10 Kbps a 500 Kbps
Transferência de Arquivos (Grandes)	10 Kbps a 1 Mbps
Vídeo (*Streaming*)	100 Kbps a 1 Mbps
Aplicação Conferência	500 Kbps a 1 Mbps
Vídeo MPEG	1 Mbps a 10 Mbps
Aplicação Imagens Médicas	10 Mbps a 100 Mbps
Aplicação Realidade Virtual	80 Mbps a 150 Mbps

Fonte: Martins *et al (* 2003).

- Atraso (*Delay*) – O atraso está associado tipicamente ao contexto da transmissão como atraso de propagação, atraso de transmissão e latência de equipamentos. De forma mais genérica é o tempo que as informações geradas em uma aplicação levam para chegar ao destino.
- Variação do atraso (*Jitter*) – A variação do atraso é causada principalmente pelo aumento da carga de tráfego nos roteadores, mas pode ser causada também por modificações de rota devido a falhas ou durante as alterações nas tabelas de rota. Outros fatores relacionados com as aplicações podem influenciar o aumento da variação do atraso (*jitter*), como sistema operacional e processos de codificação (MONTEIRO; SAMPAIO; FIGUEREDO, 2002).
- Perda de pacotes – As perdas de pacotes normalmente estão associadas principalmente a congestionamentos, mas também podem ser associadas a erros introduzidos pelos meios de transmissão.

Para redes DS-TE, há parâmetros de desempenho adicionais a serem considerados devido ao controle de admissão dos fluxos de tráfego e prioridades entre as classes de tráfegos. Dentre eles os principais são: bloqueios por classe de tráfego e total, fluxos atendidos, preempções por classe e total, carga média do enlace total e por classe de tráfego.

2.1.2 Tipos de Aplicações

Segundo Miras (2002), podem-se agrupar as aplicações em elásticas e inelásticas:

- Aplicações Elásticas (adaptativas)

As aplicações elásticas ou adaptativas toleram variações significativas na vazão e atraso sem afetar consideravelmente a sua qualidade. Nestas aplicações, quando o desempenho da rede degrada, a aplicação tende a degradar também. Este é o comportamento tradicional de aplicações de transferência de dados como transferência de arquivos, E-mail e alguns tráfegos HTTP. Situações extremas de atrasos e variação de vazão podem degradar o desempenho da aplicação, de forma a torná-la inoperante.

- Aplicações Inelásticas (tempo real)

Aplicações inelásticas, também chamadas de aplicações em tempo real, são relativamente intolerantes ao atraso, variação do atraso, vazão e erros. Geralmente oferecem suporte algum tipo de mídia sensível a QoS como voz ou comandos de controle remoto. Se certos parâmetros de QoS não são atendidos, a qualidade pode se tornar inaceitável e a aplicação pode perder a utilidade. No entanto, dependendo da função da aplicação e os tipos de mídia envolvida, o aplicativo pode operar com sucesso dentro de um intervalo de valores de QoS. Por exemplo, aplicações de fluxo (*streaming)* áudio e vídeo, sem interatividade, são normalmente tolerantes a atraso e variação do atraso (*jitter*).

Algumas aplicações inelásticas podem tolerar certos níveis de degradação QoS (aplicações tolerantes) e podem operar dentro de faixa de parâmetros de QoS com qualidade aceitável ou satisfatória. Aplicações de vídeo podem tolerar certa quantidade de perda de pacotes sem os prejuízos decorrentes, tornando-as insignificantes para o usuário. Desta forma as aplicações inelásticas por sua vez podem ser subdivididas em aplicações inelásticas tolerantes adaptativas, aplicações inelásticas tolerantes não-adaptativas e aplicações inelásticas intolerantes:

- Aplicações inelásticas tolerantes adaptativas

As aplicações inelásticas tolerantes adaptativas são capazes de suportar certos níveis de variação de atraso*,* vazão, perda de pacote e congestionamento. Este suporte é possível por meio de uso de *buffers,* redução da taxa de bits, ou por redundância nos dados enviados (por exemplo, um fluxo de vídeo pode perder alguns pacotes, quadros ou camadas). As aplicações adaptativas são, até certo ponto, capazes de ajustar suas demandas de recursos dentro de uma faixa de valores aceitáveis. A adaptação da aplicação é desencadeada por mecanismos apropriados que direta ou indiretamente informam ao aplicativo sobre desempenho atual da rede. Adaptação é também muito importante no contexto do controle

de congestionamento Internet. Um aplicativo pode adaptar-se à perda de pacotes, que é primariamente uma indicação de congestionamento da rede e, responsavelmente, reduzir a sua da taxa de transmissão.

- Aplicações inelásticas tolerantes não-adaptativas

As aplicações inelásticas tolerantes não-adaptativas podem tolerar alguma variação de QoS, mas não conseguem se adaptar às condições da rede. Por exemplo, a qualidade de um fluxo de áudio ou vídeo pode ser degradada pela perda, mas continua a ser inteligível para o usuário ouvinte ou telespectador se dentro das variações suportadas.

- Aplicações intolerantes

As aplicações intolerantes não conseguem realizar suas tarefas satisfatoriamente se suas demandas de QoS não forem atendidas. Essas aplicações são chamadas de aplicações intolerantes. Um exemplo de tal aplicação é o controle remoto de equipamentos de missão crítica, como um braço de robô ou instrumentos cirúrgicos.

Os parâmetros de qualidade de serviço associados a cada tipo de aplicação podem ser definidas em SLA (*Service Level Agreement*). Segundo Enne (2009), SLA é um contrato "entre o provedor de serviço de rede e o usuário, através do qual ficam definidos um perfil de tráfego do usuário e os compromissos de QoS por parte do provedor". Esta SLA deve definir claramente quais requisitos de Qualidade de Serviço da Rede devem ser garantidos para que a(s) aplicação(ões) possa(m) ser executadas com qualidade (TRIMINTZIOS *et al*, 2001).
Um exemplo típico de SLA para uma aplicação de voz sobre IP é indicada na tabela 2.

Tabela 2 - Exemplo Típico de SLA para Aplicação de Voz sobre IP

Parâmetro	Especificação
Perda	**≤ 10 %**
Atraso	≤ **200 ms**
Jitter	≤ **40 ms**

O grande conflito para prover QoS nas redes atuais é a herança das características do seu principal protocolo: o IP. Dentre todas as suas características, o serviço de entrega do melhor esforço (*Best Effort*) com comutação de pacotes é o maior desafio para arbitrar os diferentes requisitos das aplicações, pois determina que os dados sejam divididos em pacotes para serem

encaminhadas pela rede de forma independente, podendo percorrer caminhos distintos de outros pacotes. Este serviço não provê nenhuma garantia de banda, atraso máximo ou perdas (entrega dos pacotes no destino) (SHENKER; PARTRIDGE; GUERIN, 1995; MOREIRA *et al*, 2009).

2.2 PROVIMENTO DE QUALIDADE DE SERVIÇO EM REDES IP

Uma das formas de prover QoS em redes IP (serviço do melhor esforço) é o superdimensionamento, a solução onde se disponibilizam recursos muito além do necessário, como banda, espaço em *buffers* e capacidade nos roteadores, a fim de evitar possíveis disputas. O superdimensionamento é uma forma intuitiva e muito utilizada de obter QoS, porém deve-se levar em conta sua viabilidade financeira. Cabe ressaltar que superdimensionamento não livra de possíveis picos de congestionamento na rede (TANENBAUM, 2003).

A outra forma de prover QoS em redes IP é utilizar soluções complementares à arquitetura IP que arbitrem de forma adequada o uso de recursos da rede. Dentre as atuais e mais frequentemente utilizadas para implantação de um suporte de QoS em redes IP estão: Serviços integrados (IntServ), Serviços diferenciados (DiffServ) e MPLS/DS-TE.

2.3 SERVIÇOS INTEGRADOS (INTSERV)

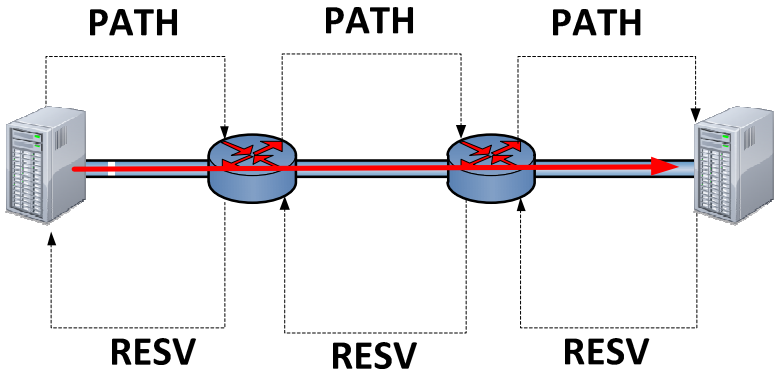
A solução IntServ (*Integrated Services*) baseia-se no esquema de reserva de recursos (banda, atraso e *jitter*) para os fluxos de tráfego antes do estabelecimento da comunicação (ARMITAGE, 2000).

O IntServ utiliza o protocolo de sinalização RSVP (*Resource Reservation Protocol*) para realizar o gerenciamento de recursos em todo o caminho por onde passará o fluxo, e com isto cria-se uma espécie de circuito virtual com recursos reservados para cada fluxo na rede (ARMITAGE, 2000; MONTEIRO; SAMPAIO; FIGUEREDO, 2002).

Na sinalização da reserva de recursos através do protocolo RSVP, há uma propagação da mensagem tipo PATH (*path establishment*) da origem do pacote por todos os elementos do caminho até o elemento de destino. Na sequência, o destino confirma a reserva através da propagação de mensagens do tipo RESV (*reservation*) pelo caminho inverso das mensagens PATH quando as reservas são de fato realizadas, como pode ser observado na

figura 3. Caso algum dos elementos do caminho não disponha dos recursos requisitados, as reservas já feitas até aquele ponto são desfeitas.

Figura 3 – Reserva de Recursos – Arquitetura IntServ (RSVP)



A arquitetura IntServ provê dois tipos de serviços além do de melhor esforço: o serviço garantido e o serviço de carga controlada.

O serviço garantido provê um nível assegurado de banda, um limite rígido de atraso fim-a-fim e uma proteção contra a perda de pacotes nas filas, para os pacotes que estiverem obedecendo ao perfil de tráfego contratado (SLA). O serviço garantido é recomendado para aplicações com requisitos rígidos como aplicações inelásticas intolerantes, pois precisam de uma garantia rígida de que um pacote não irá chegar ao destino depois de um tempo maior que um limite especificado. Este serviço não oferece garantia mínima da variação do atraso (*jitter*), uma vez que ele simplesmente garante um atraso máximo gerado pelas filas (KAMIENSKI; SADOK, 2000).

O serviço de carga controlada não oferece garantias quantitativas rígidas, como o serviço garantido. O serviço de carga controlada prevê que um alto percentual de pacotes transmitidos chegará com sucesso no destino e que o atraso sofrido por um alto percentual dos pacotes não deverá exceder tanto o atraso mínimo sofrido por um pacote dentro de um fluxo. A diferença entre este serviço e o de melhor esforço é que os pacotes sob este serviço não são deteriorados, significantemente, quando a carga da rede aumenta (KAMIENSKI; SADOK, 2000).

Segundo Monteiro, Sampaio e Figueredo (2002), o maior problema da arquitetura IntServ é a escalabilidade, pois os roteadores precisam manter os estados de cada circuito virtual que os atravessa, e à medida que aumenta a quantidade de circuitos simultâneos, aumenta também o processamento nos roteadores, o que pode gerar a chamada “explosão de estados”.

2.4 SERVIÇOS DIFERENCIADOS (DIFFSERV)

A arquitetura de serviços diferenciados, comumente chamada de DiffServ (*Differentiated Services* - DS), possui como característica principal a diferenciação dos serviços definidos no roteador para agregados de fluxos. Esses agregados de fluxos com o mesmo comportamento (*Behavior Aggregate* - BA) implantado na rede são também chamados de Classes de Serviço (CoS) na medida em que correspondem a agrupamentos de fluxos (classes) sendo tratados de forma, em princípio, homogênea na rede DiffServ (BLAKE *et al*, 1998).

2.4.1 Configuração de Recursos por Classe

A agregação dos fluxos em classes de serviço permite que a utilização dos recursos do roteador (filas e banda disponível nas interfaces) seja previamente configurada por classe de serviço, segundo um "projeto de qualidade de serviço - QoS". Dessa forma, os aplicativos não precisam fazer reserva prévia de recursos para seus fluxos de dados.

Além disso, a configuração prévia dos recursos a serem utilizados pelas classes de serviço elimina a necessidade do armazenamento dos estados de cada fluxo nos roteadores. Com isso a complexidade da rede não cresce proporcionalmente ao número de fluxos individuais levando a uma maior escalabilidade da solução.

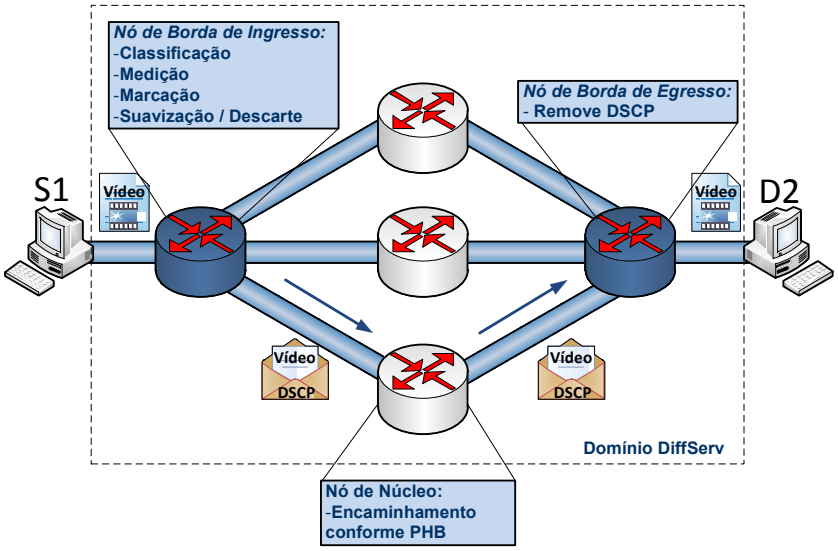
2.4.2 Nós DS de Borda e de Núcleo

Os roteadores habilitados para DiffServ são chamados nós DS (DiffServ) e suas funções são separadas em nós DS de núcleo e de borda.

Os nós DS de borda são responsáveis, como será visto a seguir, principalmente pelo condicionamento do tráfego que ingressa em um domínio DS. Nestes nós, são realizados os agrupamentos dos fluxos em *Behavior Aggregates* - BA (classes de serviço) pré-definidos. Os nós DS de borda também são responsáveis por se comunicar com nós DS de borda de outros domínios DS.

Os nós DS de núcleo comunicam-se internamente e têm a função de encaminhar o tráfego já previamente condicionado conforme os Behavior Aggregates - BA (classes de serviço) configurados pelo gerente da rede. Desta forma, a complexidade dos nós de núcleo DS é reduzida, requerendo menos recursos computacionais dos mesmos. A figura 4 ilustra um domínio DS com seus nós de núcleo e nós de borda.

Figura 4 – Domínio DS



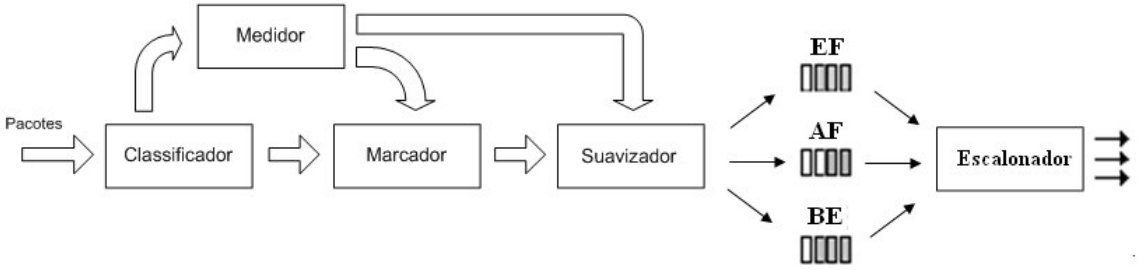
2.4.3 Condicionamento do Tráfego

As atividades referentes ao policiamento dos pacotes nos roteadores de borda para averiguar sua adequação ao perfil de tráfego contratado são coletivamente chamadas de condicionamento de tráfego (KAMIENSKI; SADOK, 2000).

O processo de condicionamento envolve a classificação, medição, marcação e suavização ou descarte dos pacotes em conformidade com o perfil de tráfego contratado (SLA).

A figura 5 mostra os possíveis estágios de um elemento condicionador de tráfego.

Figura 5 – Condicionador de Tráfego



Fonte: Pinto Neto (2008)

2.4.3.1 Classificador

O classificador é o primeiro elemento do processo de condicionamento de tráfego. Os dois principais tipos de classificadores são o classificador BA (*Behavior Aggregate*) e o classificador MF (*Multi-Field*).

O classificador BA (*Behavior Aggregate*) classifica os pacotes conforme o conteúdo do campo DSCP (*Differentiated Service Code Point*). São implementados em nós de borda, onde o domínio anterior é compatível com o DiffServ, e os pacotes já possuem marcação.

O classificador MF (*Multi-Field*) classifica os pacotes recebidos nas interfaces de entrada baseado no conteúdo de alguma parte do seu cabeçalho como: endereço do destino, da fonte e número da porta. Normalmente são implementados em nós de borda em que o domínio anterior não marca o DSCP dos pacotes.

O resultado da classificação é o enquadramento do pacote em um BA (*Behavior Aggregate*) válido no domínio e, geralmente, encaminhamento para a fase de medição. Caso o pacote esteja usando o serviço de melhor esforço, normalmente ele será encaminhado sem nenhum processamento adicional.

2.4.3.2 Medidor

O medidor afere as propriedades temporais de um fluxo de pacotes de acordo com o perfil de tráfego especificado, a fim de avaliar quais pacotes estão dentro ou fora do perfil contratado.

Segundo Kamienski e Sadok (2000), "embora vários mecanismos de medição possam ser utilizados, o mais apropriado é o balde de fichas". No sistema de balde de fichas, os pacotes fora do perfil são aqueles que chegam quando não existem fichas suficientes no balde.

Após a medição, em geral, os pacotes são encaminhados para suavização, descarte, marcação ou contabilização para posterior cobrança.

Pacotes provenientes do classificador BA dentro do perfil são encaminhados sem nenhum processamento adicional. Já os pacotes resultantes do classificador MF precisam ser encaminhados para marcação do seu DSCP (*Differentiated Service Code Point*).

2.4.3.3 Marcador

A marcação é necessária sempre que um pacote for proveniente do classificador MF ou por algum motivo precise ser remarcado. Ela é importante para que os nós de núcleo possam encaminhar os pacotes de acordo com seu *Behavior Aggregate - BA* (classe de serviço).

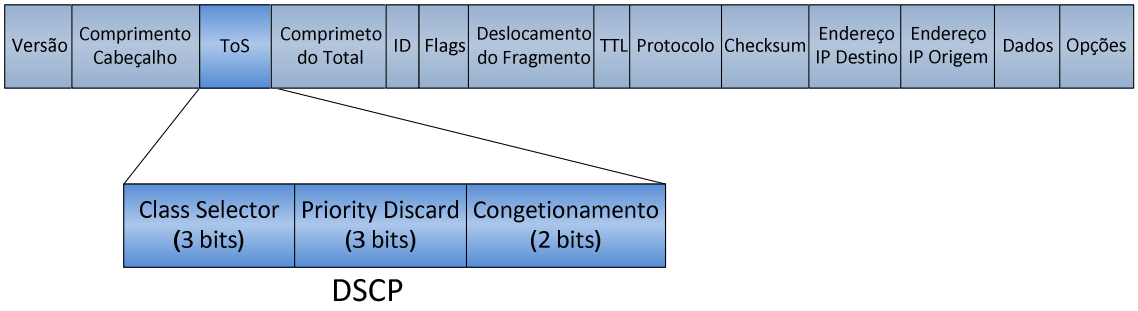
A remarcação normalmente é utilizada para rebaixar fluxos fora do perfil ou quando dois domínios DS utilizam valores do DSCP (*Differentiated Service Code Point*) diferentes para o mesmo BA.

A marcação é feita no cabeçalho do pacote IP com a atribuição de um valor (máximo de 6 bits), de acordo com o *Behavior Aggregate - BA* (classe de serviço), através do campo denominado DSCP (*Differentiated Service Code Point*). O DSCP é subdividido em duas

partes: *Class Selector* (3 primeiros bits) e *Priority Discard* (3 últimos bits). Cada *Behavior Aggregate – BA* (classe de serviço) deve possuir o mesmo identificador DSCP para um domínio DS, possibilitando com isso, que outros nós, ao longo do caminho, consigam identificar o *Behavior Aggregate – BA* (classe de serviço) do fluxo de forma simples.

No IPv4, utiliza-se o campo DS[1] (*Differentiated Service*) com os primeiros 6 utilizados para a marcação do DSCP, e os dois bits restantes para sinalização de congestionamento, como pode ser observado na figura 6. No IPv6, utiliza-se o campo "*Traffic Class*" para a marcação do DSCP.

Figura 6 – Campo DSCP



2.4.3.4 Suavizador / Descartador

O suavizador faz a moldagem do tráfego, baseado em alguma propriedade configurada e especificada na SLA. De forma geral o suavizador, introduz um atraso nos pacotes fora do perfil. Como exemplo, o tráfego pode ser moldado com base em uma taxa de pico acordada entre cliente e provedor, e os pacotes que não estiverem em conformidade podem ser descartados (MONTEIRO; SAMPAIO; FIGUEREDO, 2002).

O descartador pode ser implementado como um caso especial de condicionamento. Sua função é descartar pacotes que foram considerados fora do perfil contratado.

2.4.4 PHB (Per-Hop Behavior)

O PHB (*Per-Hop Behavior*) é a maneira como um roteador aloca recursos e trata o encaminhamento para os BAs nos nós DS. Todos os pacotes que pertencem a um mesmo BA em um Domínio DS são tratados em todos os nós pelo mesmo PHB que é identificado através do mesmo código DSCP.

---

[1] O campo DS do cabeçalho IP é composto de 8 bits e substitui o campo ToS (Type of Service) (BLAKE et al, 1998), definido na especificação original do IP (RFC-791).

Segundo Kamienski e Sadok (2000), a maneira mais simples de implementar um PHB é destinar a ele um determinado percentual de banda nos enlaces de saída. Combinando o PHB com as regras de policiamento da borda, é possível a criação de diversos serviços numa rede DiffServ.

Atualmente, há dois PHBs padronizados pelo IETF: Encaminhamento Expresso (EF - *Expedited Forwarding*) e Encaminhamento Assegurado (AF – *Assured Forwarding*). Também foi definido um PHB *default*, ou PHB BE (*Best Effort*), para o encaminhamento de tráfego de melhor esforço.

2.4.4.1 PHB AF – Encaminhamento Assegurado (*Assured Forwarding*)

O PHB AF pode ser utilizado por um Domínio DS para oferecer níveis diferentes de garantias de encaminhamento para pacotes recebidos. Sua motivação é a demanda existente nas redes atuais por encaminhamento assegurado de pacotes IP (HEINANEN *et al*, 1999).

Foram definidas quatro classes AF independentes. Cada classe corresponde a um nível de serviço distinto, com maior ou menor probabilidade de encaminhamento. Na linguagem de mercado, são conhecidas como serviço: *platinum*, ouro, prata e bronze.

Cada uma das quatro classes AF são subdivididas em três subclasses, com diferentes níveis de precedência em relação ao descarte de pacotes. O nível de precedência determina quais pacotes serão descartados primeiro pelos roteadores em caso de congestionamento.

Para cada uma das classes, são alocados recursos nos roteadores, como banda e buffers.

2.4.4.2 PHB EF - Encaminhamento Expresso (*Expedited Forwarding*)

O PHB EF define garantias mais rígidas de QoS para aplicações muito sensíveis a variações de características temporais da rede. Ele pode ser utilizado para implementar um serviço com baixa perda de pacotes, pequeno atraso, baixa variação do atraso (*jitter*) e banda garantida (JACOBSON; NICHOLS; PODURI, 1999).

Segundo Kamienski e Sadok (2000), para os usuários, esse serviço, conhecido como Premium, parece com uma “linha privativa virtual” (VLL – *Virtual Leased Line*). É o serviço indicado para encaminhar tráfego de aplicações multimídia e de tempo real em geral.

Perda de pacotes, atrasos e variações no atraso ocorrem devido à existência de filas nos roteadores. Portanto, para a implementação do PHB EF, é necessário que em cada um dos nós a taxa de saída de pacotes para este serviço seja maior que a taxa de chegada e fixada, independente dos outros tráfegos. Esse procedimento evita a formação de filas, permitindo, consequentemente, a manutenção das garantias de QoS oferecidas por esse PHB.

O DiffServ (RFC 2475) não especifica como os PHBs devem ser implementados.

Segundo Pinto Neto (2008), um exemplo de agrupamento em classes para um conjunto de aplicações em determinada rede é mostrado na Tabela 3:

Tabela 3 - Exemplo de Mapeamento em Classes de Serviço DiffServ

Classe de Serviço	Aplicação	DSCP
EF	Voz sobre IP (VoIP) Videoconferência (Vídeo Interativo)	"101110"
AF1	Vídeo MPEG Vídeo sob Demanda (Streaming) Sinalização	"001010"
AF2	Roteamento (Gerência dos Roteadores) Gerência de Redes Web - HTTP	"001100"
AF3	FTP - File Transfer Protocol Aplicações TCP (Outras)	"001110"
BE	Demais Aplicações de Dados	"010010"

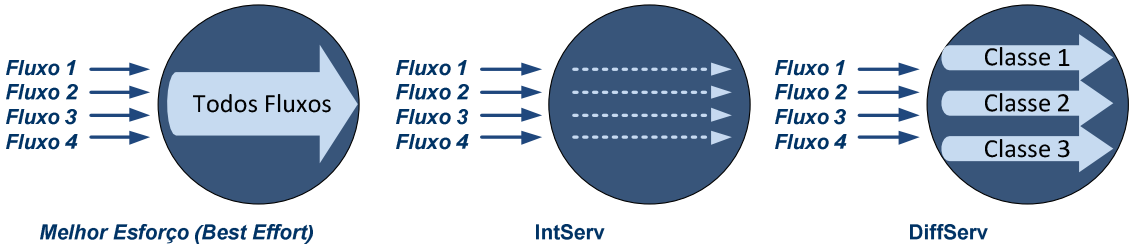
2.4.5 Considerações sobre o DiffServ

A agregação de fluxos e redução da complexidade dos nós DS permite uma maior escalabilidade, possibilitando o uso da arquitetura DiffServ em *backbones* de grandes redes.

O tratamento diferenciado de pacotes por classe de serviço possibilita à arquitetura DiffServ lidar com as diferentes necessidades (demandas) das aplicações em rede. Porém, cabe registrar que o DiffServ não oferece a garantia de recursos para todos os fluxos. Um fluxo individual pode não atingir as suas necessidades em termos dos parâmetros de QoS, como banda e atraso devido à saturação ou configuração incorreta dos recursos para os serviços da rede.

A figura 7 exemplifica de forma gráfica a diferença no tratamento dos fluxos no melhor esforço (*Best Effort*), IntServ e DiffServ.

Figura 7 – Melhor Esforço versus IntServ versus DiffServ

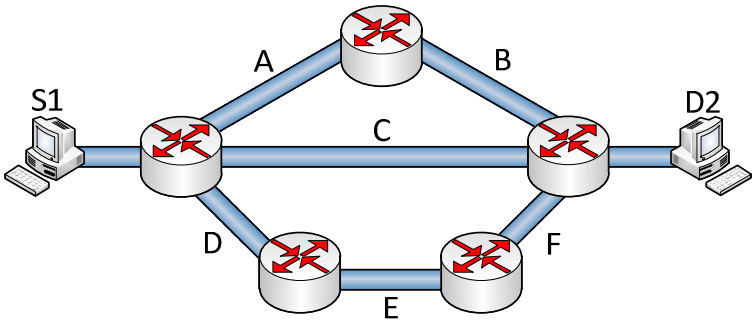


Fonte: Santana (2006).

2.5 ENGENHARIA DE TRÁFEGO

Em redes de grande porte, como aquelas das operadoras de telecomunicações e grandes corporações, é comum haver diversos caminhos (rotas) entre a origem e o destino dos dados. Muitas vezes, estas rotas são definidas de forma estática pelo gerente da rede, ou através de algoritmos de roteamento. Comumente, escolhas de caminho visam a minimizar métricas simples sem levar em conta o estado da rede como um todo. A métrica mais comumente usada para escolha de caminho é o número de saltos; mas em se tratando de QoS, o caminho mais curto pode nem sempre apresentar o melhor conjunto de recursos para a aplicação.

Figura 8 – Topologia com Diversos Caminhos

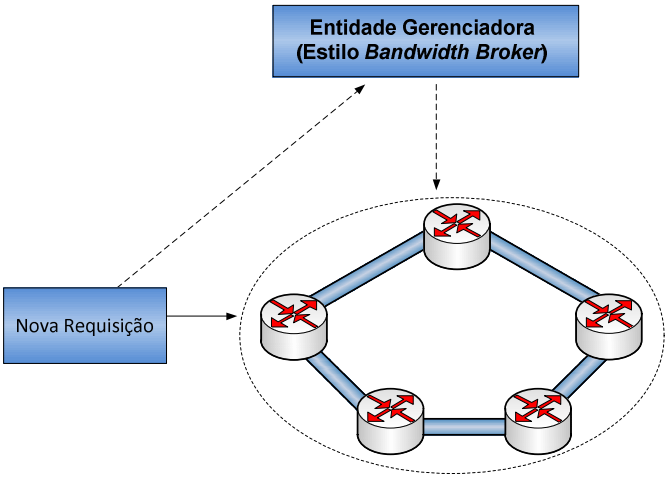


A fim de otimizar a utilização dos recursos da rede, são adotadas técnicas de Engenharia de Tráfego (ET). A Engenharia de Tráfego (ET) está preocupada com a otimização de desempenho de redes operacionais. Em geral, ela engloba a aplicação dos princípios da tecnologia e científicos para a medição, modelagem, caracterização e controle do tráfego da rede, bem como a aplicação desses conhecimentos e técnicas para atingir os objetivos de desempenho específicos. Um dos principais objetivos da Engenharia de Tráfego é facilitar a

operação da rede, de forma eficiente e confiável ao mesmo tempo, otimizando a utilização de recursos da rede e desempenho do tráfego (AWDUCHE *et al*, 1999).

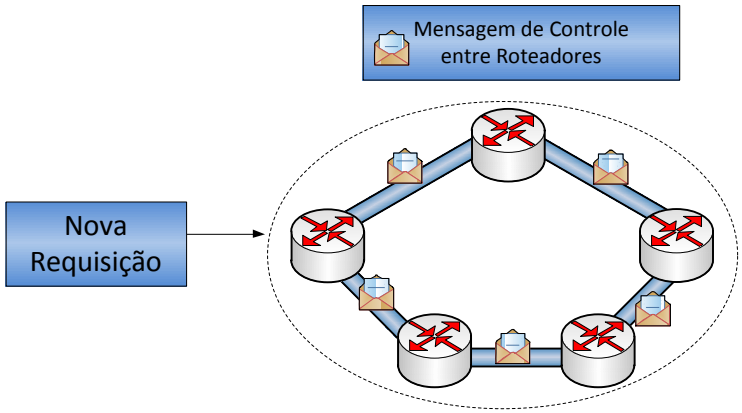
O melhor aproveitamento dos recursos de rede por técnicas de engenharia de tráfego possibilita a melhoria da qualidade de serviço da rede em geral. A Engenharia de Tráfego pode estar associada a um controle centralizado ou distribuído. No controle centralizado (figura 9), existe uma unidade gerenciadora central que tem a visão total das configurações da rede. Esta unidade gerenciadora central é responsável pela determinação das rotas e plano de roteamento além de coletar informações do estado da rede, cálculo de rotas e determinação, para todos os rotadores, das configurações de roteamento (AWDUCHE, 1999b).

Figura 9 - Gerenciamento Centralizado



Já no controle distribuído (Figura 10) cada roteador, de uma forma independente, seleciona as rotas dos fluxos que passam por si, baseados na sua visão do estado da rede. Tipicamente, a visão do estado da rede de cada roteador é construída através da troca de mensagens de controle entre os roteadores vizinhos.

Figura 10- Gerenciamento Distribuído



Tanto no controle distribuído como no controle centralizado, há necessidade de uma infraestrutura de rede que permita o controle de admissão e roteamento baseado em restrições para o suporte às técnicas de engenharia de tráfego. O controle de admissão é importante em engenharia de tráfego para que não sejam aceitas mais requisições do que a rede pode suportar, e o roteamento baseado em restrições é importante para a escolha do melhor caminho não só para o fluxo, mas também para otimizar a rede como um todo.

Segundo Trimintzios e outros (2001), em geral existem duas abordagens de engenharia de tráfego:

- A Engenharia de tráfego baseada em MPLS, que baseia-se em um paradigma de roteamento explícito, no qual um conjunto de rotas (caminhos) é calculado *offline* para tipos específicos de tráfego. Além disso, recursos de rede apropriados, por exemplo, a banda, podem ser provisionada ao longo das rotas de acordo com as necessidades de tráfego previsto. O tráfego é roteado de dianamicamente, dentro do conjunto de rotas estabelecidas e de acordo com o estado da rede.
- A Engenharia de tráfego baseada em IP baseia-se em uma estratégia de roteamento "liberal", onde as rotas são calculadas de forma distribuída, descobertas pelos próprios roteadores. Embora a seleção de rotas seja realizada de forma distribuída, as decisões de roteamento baseado em QoS são restringidas de acordo com as considerações de engenharia tráfego para toda a rede. Os algoritmos de roteamento dinâmico atribuem dinamicamente as métricas de custo para cada interface de rede. O cálculo de rota é normalmente baseado em algoritmos de seleção de

menor caminho. A fim de permitir que as rotas sejam computadas por tipo de tráfego ou de classe, podem ser alocados vários custos a um enlace.

Ainda segundo Trimintzios e outros (2001), a engenharia de tráfego baseada em MPLS é exercida em duas escalas de tempo, em longo e em curto prazos:

- Longo prazo (dias-semanas) – Seleciona o tráfego que será encaminhado pelo MPLS baseando-se em cargas de tráfego previsto e SLAs de longo prazo existentes. Os caminhos são roteados explicitamente, assim como o agendamento dos roteadores associados e os mecanismos de buffer são definidos *offline*.

- Curto prazo (minutos, horas) – Baseia-se na observação do estado operacional da rede. Alocação dinâmica de recursos e procedimentos de gerenciamento de rotas são empregados de modo a garantir a alta utilização de recursos e equilibrar o tráfego de rede entre os caminhos roteados explicitamente pela engenharia de tráfego de longo prazo.

Este trabalho utiliza como infraestrutura para suporte a QoS a Engenharia de Tráfego baseada em MPLS, de acordo com a referência DS-TE (*DiffServ Aware MPLS Traffic Engineering*) proposta por Le Faucher e Lai (2003). Esta escolha baseia-se no fato de que as redes DS-TE possibilitam o mapeamento dos fluxos conforme suas restrições de QoS (roteamento baseado em restrições), agrupados em classes de tráfego (CT), alocando recursos necessários fim-a-fim (LSP-MPLS). Também é feito um controle de admissão através dos seus modelos de alocação de banda que definem as regras a serem utilizadas na reserva de banda por CT e a aceitação de novos fluxos (LSP-MPLS) nos enlaces.

Devido à escolha das redes DS-TE como base a este trabalho, apresentaremos no próximo capítulo a tecnologia MPLS (*Multiprotocol Label Switching*), sua aplicação para engenharia de tráfego denominado MPLS-TE (*Multiprotocol Label Switching for Traffic Engineering*) e por fim as redes DS-TE.

2 REDES DS-TE (DIFFSERV AWARE MPLS TRAFFIC ENGINEERING)

Neste capítulo, apresentaremos inicialmente a tecnologia MPLS (*Multiprotocol Label Switching*) e o seu uso para aplicação de engenharia denominada MPLS-TE (*Multiprotocol Label Switching for Traffic Engineering*) para, em seguida, apresentarmos a separação de recursos em classes de tráfego denominado DS-TE, que são a base tecnológica do modelo de alocação de banda proposto (ENNE, 2009).

2.6 MPLS

Com o crescimento da Internet e das redes IP, o roteamento de pacotes passou a ser uma tarefa crítica. A cada roteador, os cabeçalhos dos pacotes IP precisam ser lidos e, com base na sua tabela de rotas, encaminhado para uma determinada interface de saída (*Table Lookup*). Essa tarefa é mais crítica para roteadores de núcleo da rede onde estas tabelas podem crescer em tamanho e, além disso, o tráfego de pacotes tende a aumentar. Motivado a reduzir a análise e processamento do cabeçalho IP nos roteadores e assim melhorar o desempenho do roteamento, surgiram as redes MPLS (*Multiprotocol Label Switching*) (ENNE, 2009).

Com o uso de rótulos (*labels*), com a função de agregar fluxos que possuem as mesmas necessidades de encaminhamento, uma rede habilitada a MPLS transforma a tarefa de roteamento de pacotes em uma simples atividade de comutação de pacotes, através de uma busca indexada em uma tabela de rótulos. Apesar do foco das redes MPLS ser as redes IP, o cabeçalho MPLS, que contém o rótulo, é colocado entre os cabeçalhos da camada de enlace e camada de rede, permitindo-as trabalhar de forma independente dos protocolos destas. Devido à característica de independência da camada de enlace e de rede nas redes que o utilizam,, o MPLS é definido como multiprotocolo (*multiprotocol*).

As redes MPLS utilizam dois tipos de roteadores: os LERs (*Label Switch Edge Routers*) e os LSRs (*Label Switch Routers*). Os LERs são roteadores de borda que têm por objetivo ler os cabeçalhos IP e designar um rótulo que será utilizado pelos LSRs, com o fim de encaminhar os pacotes em um domínio MPLS. Os LSRs são roteadores de núcleo que utilizam apenas os rótulos como índice para uma busca indexada em sua tabela de informação de encaminhamento por rótulo (LIB), sendo esta a única informação necessária para o encaminhamento dos pacotes (*forwarding*). Cabe enfatizar que os LSRs não precisam analisar o cabeçalho dos pacotes IP para encaminhá-los.

O resultado operacional das redes MPLS são caminhos comutados por rótulos (*label*) chamados LSPs (*Label Switched Paths*), que tornam as redes MPLS também uma alternativa

tecnológica para engenharia de tráfego, devido ao encaminhamento de pacotes poderem ser orientados a uma escolha da trajetória fim-a-fim (*path*) e através da utilização do mecanismo IP de roteamento explícito na fonte (*IP Source Routing*). Esta facilidade é explorada pelo MPLS-TE (*MPLS for Traffic Engineering*) apresentada adiante.

2.6.1 Elementos de uma Rede MPLS

Uma rede MPLS é composta pelos seguintes elementos que passaremos a descrever:

- Cabeçalho MPLS;
- *Label Switched Path* (LSP) – Caminho comutado por rótulo;
- *Forwarding Equivalency Class* (FEC) – Classe de equivalência de encaminhamento;
- *Label Switch Router* (LSR) – Roteador de núcleo de uma rede MPLS;
- *Label Edge Router* (LER) – Roteador de borda de uma rede MPLS;
- *Label Information Database* (LIB) ou *Label Switching Forward Tables* - Tabelas de informação de encaminhamento por rótulo.

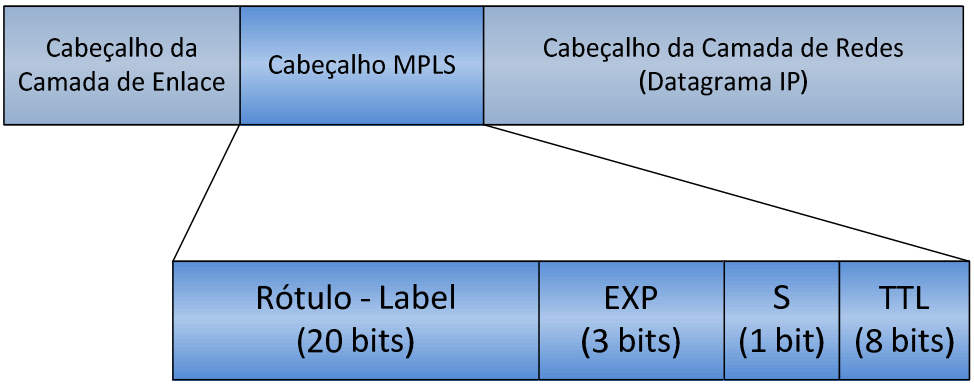
2.6.1.1 Cabeçalho MPLS

Existem duas formas de codificação de rótulos MPLS (*labels*):

- Rótulos codificados em campos já existentes nos protocolos da camada de enlace que suportam o MPLS como o ATM e *Frame Relay*.
- Rótulos codificados no interior de *shim label headers,* denominado codificação genérica (*generic encoding).*

Na codificação genérica, o cabeçalho MPLS (shim label header), como pode ser observado na figura 11,é inserido entre os cabeçalhos de enlace e de rede pelos roteadores de borda MPLS (LER) quando o pacote entra em um domínio MPLS. Sendo retirado também por um LER, ao pacote sair do domínio MPLS (ROSEN; VISWANATHAN; CALLON, 2001).

Figura 11- Cabeçalho MPLS



O cabeçalho genérico MPLS é composto por 4 campos:

- Rótulo (20 bits) – Campo reservado para o rótulo MPLS. Este rótulo tem a função de agregar fluxos com as mesmas características de encaminhamento e é utilizado como índice na busca indexada na LIB.
- Exp (3 bits) – Utilizado para diferenciar pacotes quanto à classe de serviço ou prioridade.
- S-Stack (1 bit) – Sinaliza o último cabeçalho MPLS, caso seja utilizado o empilhamento de rótulos.
- TTL (8 bits) – O tempo de vida possui a função de evitar que pacotes fiquem rodando na rede, em laço, de forma infinita (mesma função do TTL do pacote IP).

Nas redes ATM e redes Frame Relay, os rótulos (*labels*) são codificados nos campos VPI/VCI e DLCI, respectivamente. Vale resaltar que existem situações em que os cabeçalhos genéricos (*shim label header*) são utilizados complementarmente à existência de rótulos codificados nestes campos (ENNE, 2009).

2.6.1.2 *Label Switch Router* (LSR)

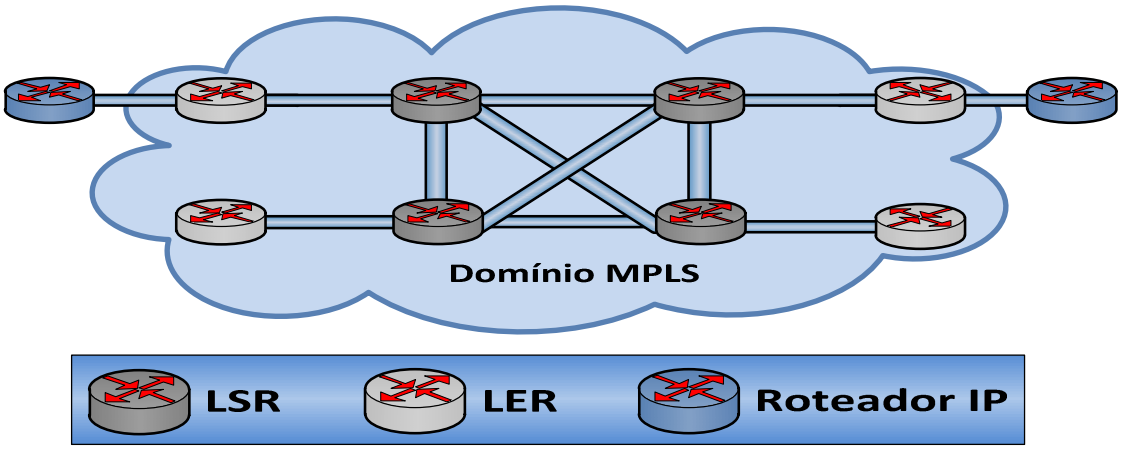
Os *Label Switch Routers* (LSRs), como podem ser observados na figura 12, são os roteadores de núcleo das redes MPLS. Os LSR são responsáveis por executar o encaminhamento baseado em rótulos e manter as tabelas de informação de encaminhamento por rótulo (LIB). Os LSRs de um domínio MPLS comunicam-se através de um protocolo de sinalização para manter atualizada sua LIB. Este processo é feito através da troca de informação (rótulos, interfaces, IPs, etc) entre LSRs vizinhos com auxílio de protocolos como LDP (*Label Distribution Protocol*) ou RSVP (*Resource Reservation Protocol*). Os LSRs encaminham

seus pacotes, com base apenas em uma busca indexada na LIB, passando como índice a interface e o rótulo de entrada.

2.6.1.3 *Label Edge Router* (LER)

Os *Label Edge Routers* (LERs), como podem ser observados na figura 12, são roteadores de borda MPLS. Além das funções de encaminhamento, os LSRs são responsáveis pela inserção do rótulo MPLS, de acordo com a classe de equivalência de encaminhamento (FEC), na entrada de um pacote em um domínio MPLS e sua retirada quando da sua saída do domínio a fim de ser entregue a uma rede não MPLS.

Figura 12 - Roteadores MPLS



2.6.1.4 *Forwarding Equivalency Class* (FEC) e *Label Switched Paths* (LSPs)

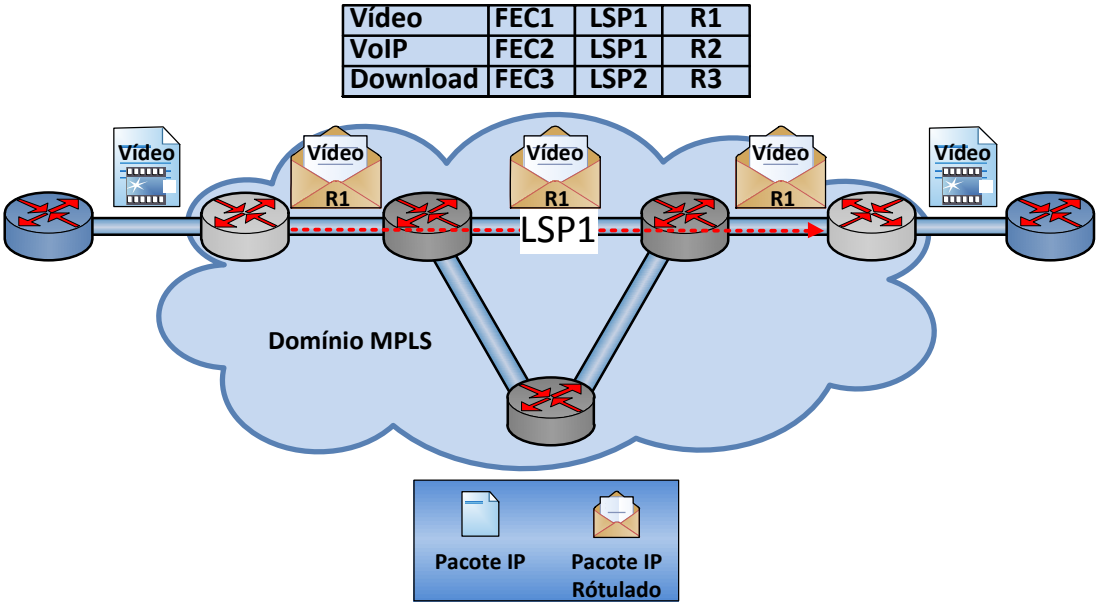
Todo pacote ao chegar a um domínio MPLS é mapeado em uma classe de equivalência de encaminhamento (FEC) no roteador de borda MPLS (LER). O mapeamento dos pacotes às FECs é definido pelo gerente da rede. O gerente da rede conta com uma gama de possibilidades para mapear os pacotes nas FECs como: endereço de destino, endereço de origem, porta de destino, porta de origem, classe de serviço, entre outros parâmetros. As FECs agrupam pacotes que terão o mesmo tratamento de encaminhamento ao longo de um domínio MPLS. Uma FEC é mapeada em um rótulo inicial que corresponde a um LSP e com isso todos os pacotes associados a uma FEC seguirão este caminho.

Os LSPs (*Label Switched Paths*) são caminhos virtuais fim-a-fim (*Path-LSP*) construídos através da comutação dos rótulos nos roteadores MPLS[2]. Todos os fluxos mapeados em uma

[2] Estes caminhos são construídos pelo plano de controle com ajuda de protocolos de sinalização.

determinado LSP seguem o mesmo caminho recebendo o mesmo encaminhamento pela rede. Devido a estas características, os LSPs são mapeados pelas FECs para encaminhar pacotes que possuem as mesmas necessidades de encaminhamento fim-a-fim (requisitos de operação). A figura 13 representa um fluxo de vídeo que foi mapeado na FEC1 e, consequentemente, associado ao LSP1.

Figura 13 - Mapeamento IP x FEC x LSP x Rótulo



2.6.1.5 *Label Information Base* (LIB) ou *Label Switching Forward Tables*

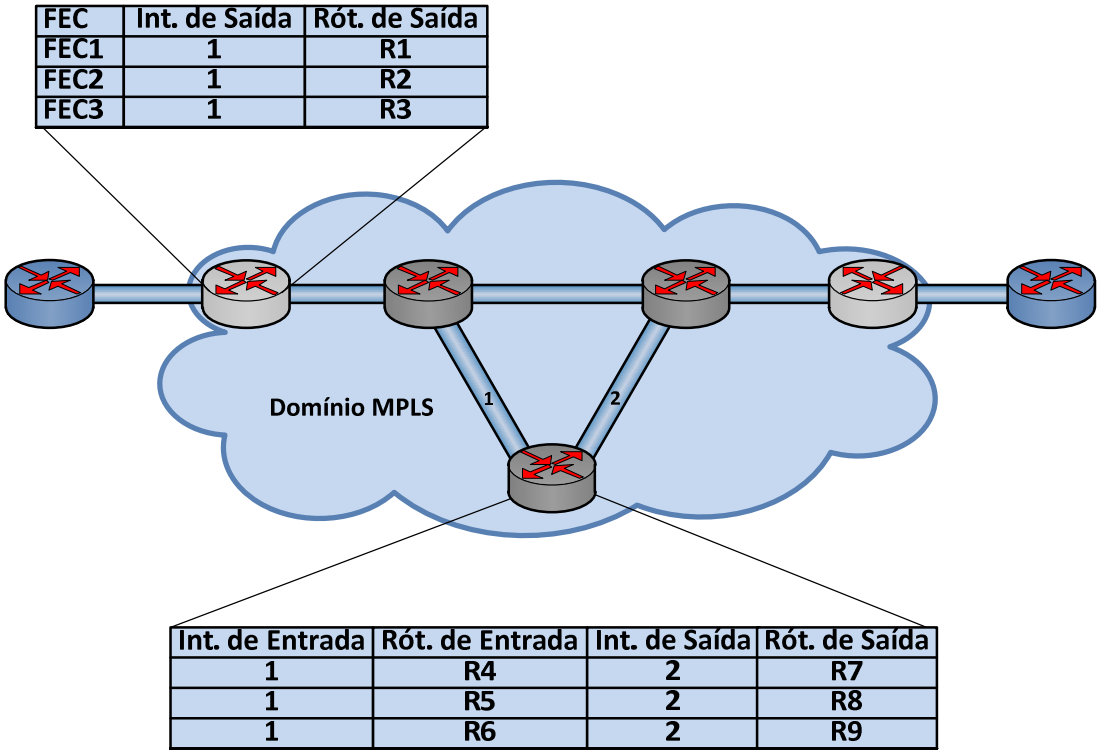
A base de informação de rótulos (LIB - Label Information Base)[3] é uma tabela de comutação de pacotes composta por registros, que associam um rótulo e interface de entrada com outra interface e um rótulo de saída para o pacote. Nos roteadores de borda MPLS (LER), a LIB associa uma interface e uma FEC de entrada com outra interface e rótulo de saída. Cada roteador com MPLS ativo possui uma LIB. As LIBs podem ser construídas manualmente pelo gerente de rede, ou dinamicamente pelo plano de controle MPLS. As LIBs são de fundamental importância no processo de encaminhamento de pacotes realizado pelo plano de encaminhamento. Encontram-se, na literatura, autores que sugerem a existência de subtabelas

[3] Também conhecida como tabela de informação de encaminhamento por rótulo.

de rótulos no roteador, as quais, por um processo de montagem, formariam a LIB. A existência destas subtabelas de rótulos não é verificado em sistemas reais.

A figura 14 ilustra as LIB nos *Label Edge Routers* (LERs) e *Label Switch Routers* (LSRs).

Figura 14 - Tabelas de Informação de Encaminhamento por Rótulo



A tecnologia MPLS pode ser dividida em dois planos: o plano de encaminhamento de pacotes (*data plane*) e o plano de controle (*control plane*). A separação destes dois planos permite uma grande flexibilidade na utilização da tecnologia MPLS.

2.6.2 Plano de Encaminhamento

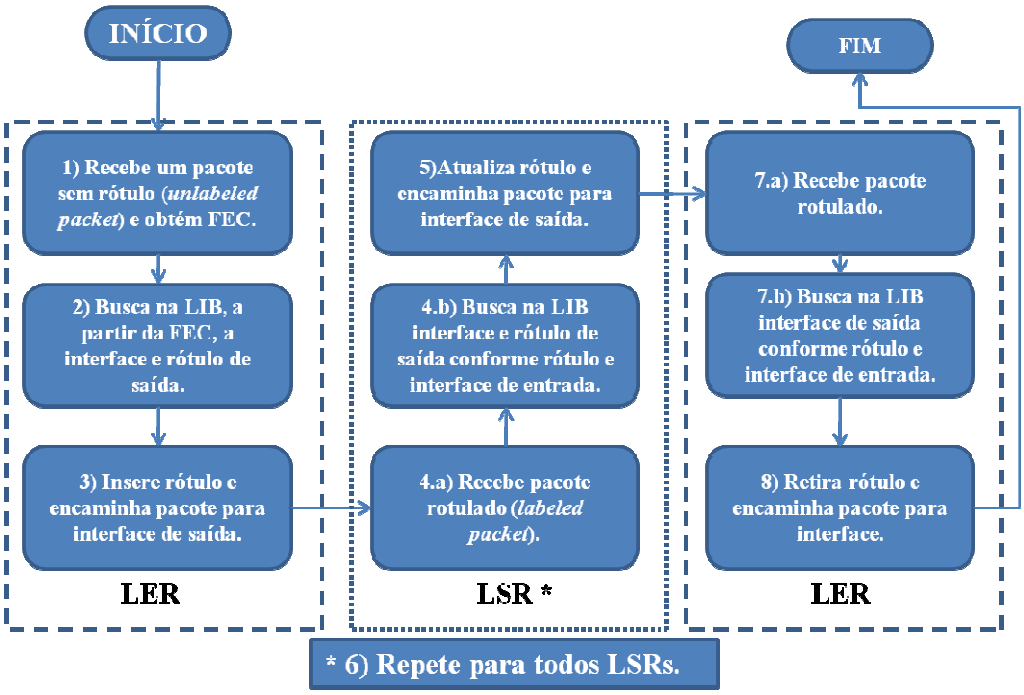
O plano de encaminhamento é responsável pelo envio de um pacote para uma determinada interface de saída. Neste plano são inseridos os cabeçalhos MPLS contendo os rótulos entre os cabeçalhos dos protocolos de enlace e de rede. A inserção do rótulo é realizada conforme a FEC definida na inspeção do cabeçalho IP nos LERs (*Label Edge Routers*). Não são necessárias novas inspeções do cabeçalho do protocolo IP no domínio MPLS. Cabe enfatizar que todo o processo de encaminhamento dos pacotes, após a inserção do primeiro rótulo, é feito pela comutação (*switch*) dos rótulos ao longo dos roteadores MPLS (*Label Switch*

Routers - LSRs) através de buscas indexadas na tabela de informação de encaminhamento por rótulo (LIB).

O processo de encaminhamento de pacotes pode ser resumido (fluxograma da figura 15) da seguinte forma:

1. O roteador de borda (LER) recebe um pacote sem rótulo (*unlabeled packet*) que entra no domínio MPLS, normalmente um datagrama IP, e verifica sua classe de equivalência de encaminhamento (FECs) a partir do prefixo do endereço IP e eventualmente outros elementos.
2. O LER busca de forma indexada em sua tabela de informação de encaminhamento por rótulo (LIB) qual o rótulo deve ser inserido no cabeçalho MPLS e qual será a interface de saída para a qual o pacote será encaminhado.
3. O LER insere o rótulo e encaminha para interface de saída.
4. Os roteadores de núcleo (LSR), ao receber um pacote MPLS (*labeled packet*), fazem uma busca indexada na LIB, passando a interface e o rótulo de entrada do pacote, a fim de encontrar o registro que possui o novo rótulo a ser inserido, além da interface para a qual o mesmo será encaminhado.
5. O LSR atualiza o rótulo e encaminha para interface de saída.
6. Os passos 4 e 5 são repetidos em todos LSR do caminho do LSP até chegar a um LER de saída do domínio MPLS.
7. O LER de saída recebe o pacote MPLS e busca em sua LIB a interface de saída, conforme rótulo e interface de entrada.
8. O LER de saída do domínio MPLS retira o cabeçalho MPLS e encaminha o pacote IP para uma interface de saída conforme a operação do IP normal.

Figura 15 - Fluxo de Encaminhamento de Pacotes MPLS



2.6.3 Plano de Controle

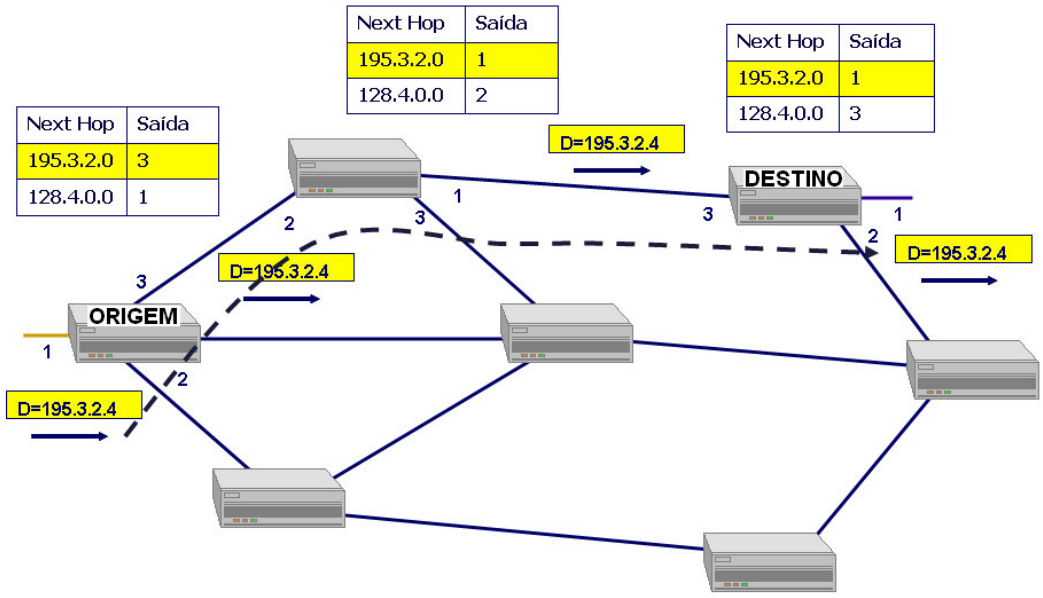
O plano de controle em uma rede MPLS é responsável por construir e manter as tabelas de informação de encaminhamento por rótulo (LIB) em todos os roteadores do domínio MPLS. Este plano efetivamente cria os LSPs através de troca de mensagens entre os roteadores MPLS ativos.

Nas redes onde o uso do MPLS é apenas para otimização do desempenho dos roteadores, a ação do plano de controle limita-se à construção e manutenção da LIB para o seu futuro uso pelo plano de encaminhamento. Neste sentido, a criação de um LSP resume-se à troca de mensagens e procedimentos entre os LSRs (alimentar as LIBs), a fim de estabelecer um caminho comutado por rótulos (*label*) conforme sua tabela de roteamento IP.

A seguir, resumem-se as etapas de atuação do plano de controle em um domínio MPLS onde se deseja apenas otimizar o desempenho do encaminhamento de pacotes:

1. Um protocolo de roteamento IP convencional (OSPF, BGP, etc) é utilizado para a construção da tabela de roteamento dos LSRs (figura 16). Estas tabelas devem conter a informação do próximo salto a ser seguido por um pacote conforme seu destino.

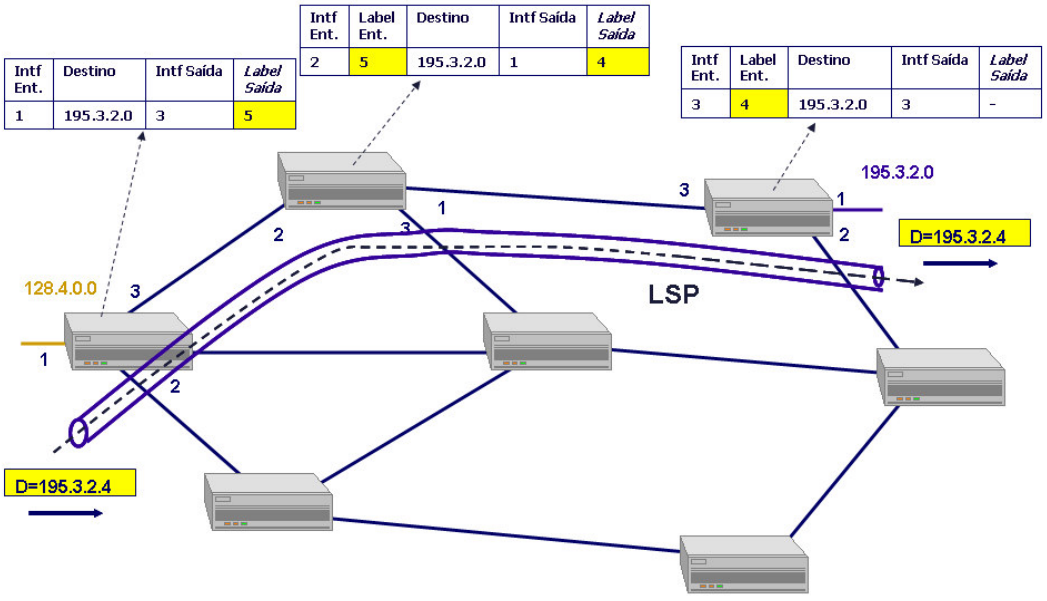
Figura 16 – Construção da Tabela de Roteamento IP convencional



Fonte: Pinto Neto (2008).

A partir das informações contidas nas tabelas de roteamento IP convencionais (figura 16), um protocolo de distribuição de rótulos, como o LDP, (*Label Distribution Protocol*) alimenta a LIB dos LSRs no caminho do novo LSP, como pode ser observado na figura 17.

Figura 17- Distribuição de Rótulos e Construção da LIB



Fonte: Pinto Neto (2008).

Devido à sua flexibilidade e à separação clara entre os planos de controle e de encaminhamento as redes MPLS são aplicadas como suporte a diversas outras aplicações. Nas próximas seções deste capítulo, abordaremos a aplicação de MPLS nas áreas de engenharia de tráfego e QoS através das recomendações MPLS-TE (*Multiprotocol Label Switching for Traffic Engineering*) e DS-TE (*DiffServ Aware MPLS Traffic Engineering*).

2.7 MPLS-TE, CÁLCULO DE ROTAS E PROTOCOLO DE SINALIZAÇÃO

Um MPLS-TE trata-se de uma rede a operar segundo os princípios básicos do MPLS, nos quais a seleção de caminhos (*LSP-Paths*) pode ser baseada em restrições.

As redes MPLS-TE são mais apropriadas para a implantação de soluções de otimização da operação de *backbones* definidas utilizando-se diferentes técnicas de engenharia de tráfego.

A implantação e operacionalização de uma rede MPLS-TE para engenharia de tráfego implica na adoção de um arcabouço de componentes com diferentes configurações possíveis. Alguns destes componentes são: a seleção de caminhos baseada em restrições, o controle de admissão, o conhecimento de estado dos enlaces, a sinalização para criação de LSPs, além do MPLS propriamente dito.

Como citado no capítulo anterior, um dos objetivos da Engenharia de Tráfego é uma melhor utilização da capacidade real da rede. Para isto é necessário um mapeamento adequado dos fluxos de pacotes de forma que sejam atendidas não apenas os requisitos de QoS de um fluxo, mas também a otimização da rede como um todo. O atendimento deste objetivo pode permitir uma melhor utilização da rede e um melhor desempenho das aplicações com a possibilidade de se evitar um superdimensionamento das larguras de banda dos enlaces da rede.

Para uma melhor utilização da capacidade da rede, há a necessidade de um suporte tecnológico que permita o controle de admissão e a seleção de caminhos baseada em restrições. As técnicas de engenharia de tráfego, não discutidas neste trabalho, usam este tipo de infraestrutura como suporte. O controle de admissão é importante em engenharia de tráfego para que não se aceitem mais requisições do que a rede possa suportar e não se degrade a qualidade de serviço como um todo. E a seleção de caminhos baseada em restrições é importante para a escolha do melhor caminho não só para os fluxos, como também para otimizar a rede como um todo. Outros aspectos complementares que a infraestrutura de rede precisa prover são: o conhecimento do estado dos enlaces e o estabelecimento dos caminhos com a possibilidade de reserva de recursos.

Em uma rede MPLS-TE, a seleção de caminho baseada em restrições, controle de admissão, conhecimento do estado dos enlaces e o estabelecimento ou sinalização do caminho são

providos por extensões ao plano de controle da arquitetura MPLS. Desta forma, detalharemos nas subseções seguintes estas extensões.

2.7.1 Informação do Estado da Rede e Cálculo de Rotas (Paths) para LSPs

Para a escolha do melhor caminho para um fluxo de pacotes é necessária a especificação de critérios passíveis de utilização para escolha deste caminho, também conhecido como requisito ou restrição.

Na tabela abaixo, são exemplificadas algumas possíveis restrições para o cálculo do caminho de um LSP (*LSP Path*).

Tabela 4 - Restrições Possíveis para Cálculo de Rotas de LSPs

Banda	Mínimo de 100Mbps da origem S1 até o destino R1.
Número de Saltos (*hops*)	Máximo de 3 saltos da origem S2 até o destino R2.
Prioridade da LSP	Caminho com banda disponível ou que tenha prioridade menor do que o novo LSP.
Perda de Pacotes	Caminho entre S1 e R2 com perda de pacotes inferiores a 0,1%.
Atraso	Caminho entre S4 e R5 com atraso inferior a 150ms.
Atributos administrativos (cor restritiva, caminho disjuntos, etc)	Caminho entre S1 e R3 que não utilize o enlace 3.

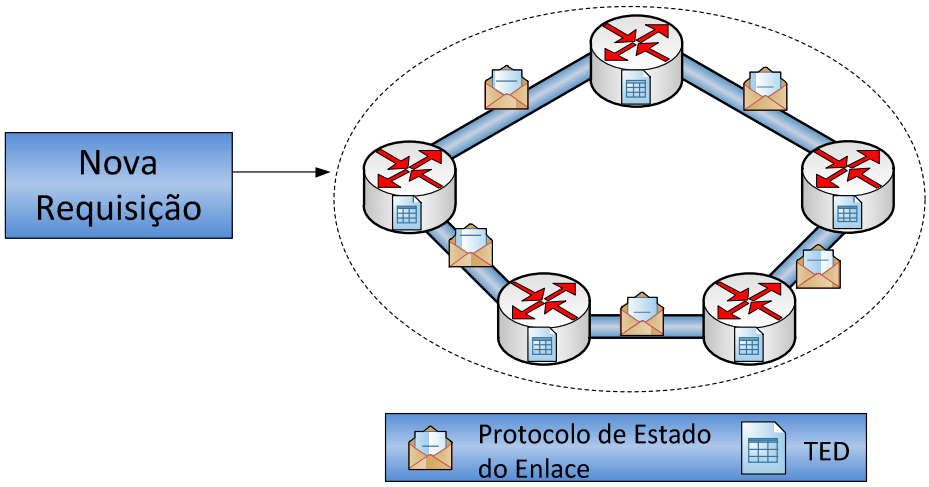
Para selecionar o caminho que atenda às restrições ou requisitos impostos a um novo LSP, é necessário que haja uma base dados com as informações da rede que serão utilizadas como entrada para o algoritmo de cálculo de caminhos baseado em restrições. Esta base de dados pode ser centralizada ou distribuída e faz parte do arcabouço de componentes normalmente utilizados numa rede MPLS-TE.

Em seguida, identificamos as duas opções básicas para o cálculo de rotas (*path*) para LSPs utilizados nas redes MPLS-TE: o controle distribuído e o controle centralizado.

No cálculo de LSPs em **controle distribuído,** existe uma base de dados de Engenharia de Tráfego denominada TED (*Traffic Engineering Database*), que está distribuída nos roteadores da rede, como pode ser observado na figura 18. Estes roteadores são os responsáveis pelo cálculo do caminho que cada LSP seguirá na rede. Para que o cálculo de rota do LSP seja efetuado de forma correta, as informações de restrições e do estado dos enlaces devem estar presentes e, também, devem ser atualizadas nos roteadores. Para isso, são necessários protocolos de estado do enlace como o ISIS-TE (*Intermediate System-to-Intermediate System - Traffic Engineering*) ou OSPF-TE (*Open Shortest Path First - Traffic Engineering*) que,

periodicamente, atualizam a TED em cada roteador e procuram garantir o cálculo de rota do LSP com informações atualizadas (LE FAUCHER, 2005b; OSBORNE; SIMBA, 2003).

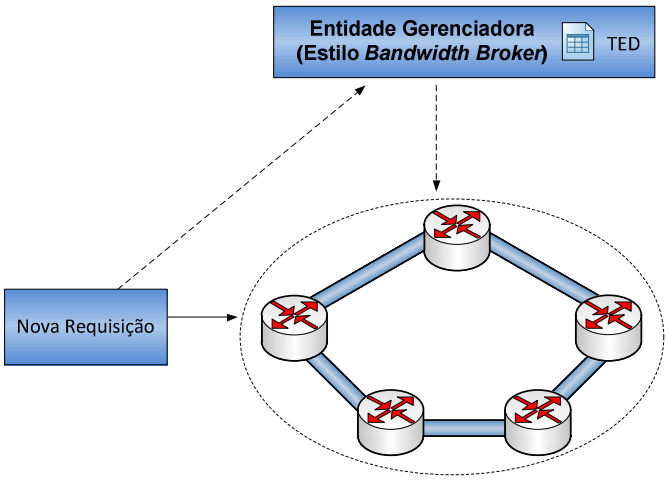
Figura 18 – Cálculo de LSPs com Controle Distribuído Arcabouço MPLS-TE



Para que não haja uma inundação de anúncios de informações do estado da rede, a propagação das informações através dos protocolos de estado de enlace é feita periodicamente, tipicamente com intervalo de 30 minutos, com novas propagações feitas quando houver alterações na banda disponível dos enlaces ou outras alterações consideradas relevantes (MINEI; LUCEK, 2005).

No cálculo de LSPs com **controle centralizado**, uma entidade gerenciadora central realiza o cálculo da rota e aciona o protocolo de estabelecimento de LSPs do plano de controle. Nesta entidade gerenciadora central, serão concentradas as informações dos estados da topologia e enlaces da rede. A entidade gerenciadora central deve ser capaz de receber uma requisição de estabelecimento de LSP do usuário, selecionar o caminho baseado em suas restrições, enviar comandos e configurações aos roteadores da rede de forma que seja possível o estabelecimento do novo LSP, preemptar LSPs e/ou re-rotear o tráfego de acordo com as necessidades da rede (SCOGLIO *et al*, 2004).

Figura 19 – Cálculo de LSPs com Controle Centralizado Arcabouço MPLS-TE



2.7.2 Cálculo de Rotas para LSPs

A escolha do caminho em uma rede MPLS-TE normalmente precisa ir além de simplesmente otimizar uma única métrica como número de saltos ou banda disponível. É necessário, também, considerar as restrições definidas pela gerência de rede. Assim sendo, é preciso que ao mesmo tempo esse caminho não viole um conjunto de restrições em consonância com os requisitos de QoS do LSP, bem como dos requisitos de otimização da rede como um todo, configurado pelo gerente da rede.

Os algoritmos de cálculos de caminho destinados a encontrar uma rota para o LSP que otimize certa métrica e, ao mesmo tempo, não viole um conjunto de restrições, são denominados algoritmos de cálculos de caminho baseados em restrições.

A engenharia de tráfego, normalmente, utiliza diferentes tipos de algoritmos de cálculo de caminhos. Segundo Pinto Neto (2008), um algoritmo especificado para o cálculo de caminho baseado em restrições para redes que utilizam o MPLS-TE é o CSPF (*Constrained Shortest Path First*) por considerar apenas os caminhos que satisfaçam as restrições pré-definidas pelo gerente da rede para o cálculo (MINEI; LUCEK, 2005).

2.7.3 Estabelecimento do LSP

Após a escolha do caminho do novo LSP, é necessário o efetivo estabelecimento do mesmo conforme as restrições configuradas. O estabelecimento do LSP também é executado pelos protocolos de sinalização tais como: LDP-CR (*Constraint-based Routing Label Distribution Protocol*), RSVP-TE (*Resource Reservation Protocol - Traffic Engineering*) e BGP-TE

(*Border Gateway Protocol - Traffic Engineering*). O processo de estabelecimento do LSP no MPLS-TE difere do usado no MPLS pelo caminho ser informado explicitamente pelo algoritmo de seleção de caminhos baseado em restrições (MINEI; LUCEK, 2005).

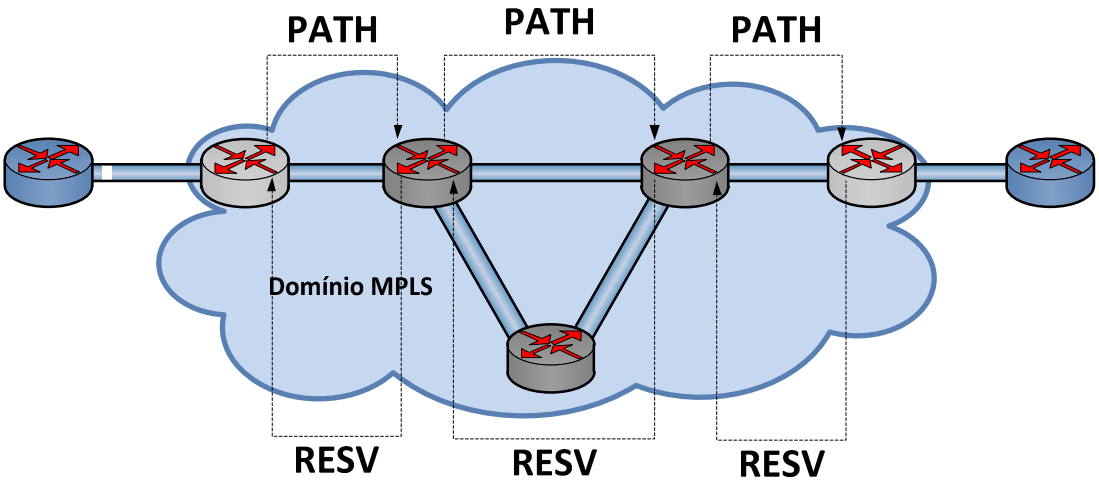
Os dois protocolos mais difundidos para o estabelecimento de caminhos em redes MPLS-TE são os protocolos RSVP-TE (*Resource Reservation Protocol - Traffic Engineering* – RFC 3209) e CR-LDP (*Constraint-based Routing Label Distribution Protocol* – RFC 3212) (AWDUCHE *et al*, 2001; JAMOUSSI *et al*, 2002).

O CR-LDP é uma evolução do protocolo LDP (*Label Distribution Protocol*) que além de distribuir os rótulos entre os roteadores MPLS com intuito de estabelecer novos LSPs permite especificar a reserva de banda nos enlaces pelos quais o novo LSP irá passar. Em fevereiro de 2003, o IETF definiu na RFC 3468 o protocolo CR-LDP como obsoleto, concentrando esforços exclusivamente no protocolo RSVP-TE. Devido a este fato focaremos o estabelecimento e controle de admissão deste trabalho no protocolo RSVP-TE (ANDERSSON; SWALLOW, 2003).

O protocolo RSVP-TE é uma extensão do protocolo RSVP (*Resource Reservation Protocol*) que tem por objetivo a configuração, manutenção, encerramento e sinalização de erro dos caminhos MPLS-TE (LSPs).

A operação básica do RSVP-TE para o estabelecimento de um LSP em uma rede MPLS é feita através da troca de mensagens dos tipos PATH e RESV. Após a escolha do caminho, pelo algoritmo de seleção de caminhos baseado em restrições, o plano de controle envia uma solicitação de estabelecimento para o roteador de borda MPLS (LER) de origem e este propaga mensagens do tipo PATH para o próximo roteador do caminho selecionado, que por sua vez para o próximo até chegar ao roteador de borda MPLS (LER) de destino. O roteador de destino confirma a reserva de recursos através de mensagens RESV, que são propagadas até o roteador de origem como pode ser observado na (figura 20) (AWDUCHE *et al*, 2001).

Figura 20 - Troca de Mensagens PATH e RESV no Protocolo RSVP-TE

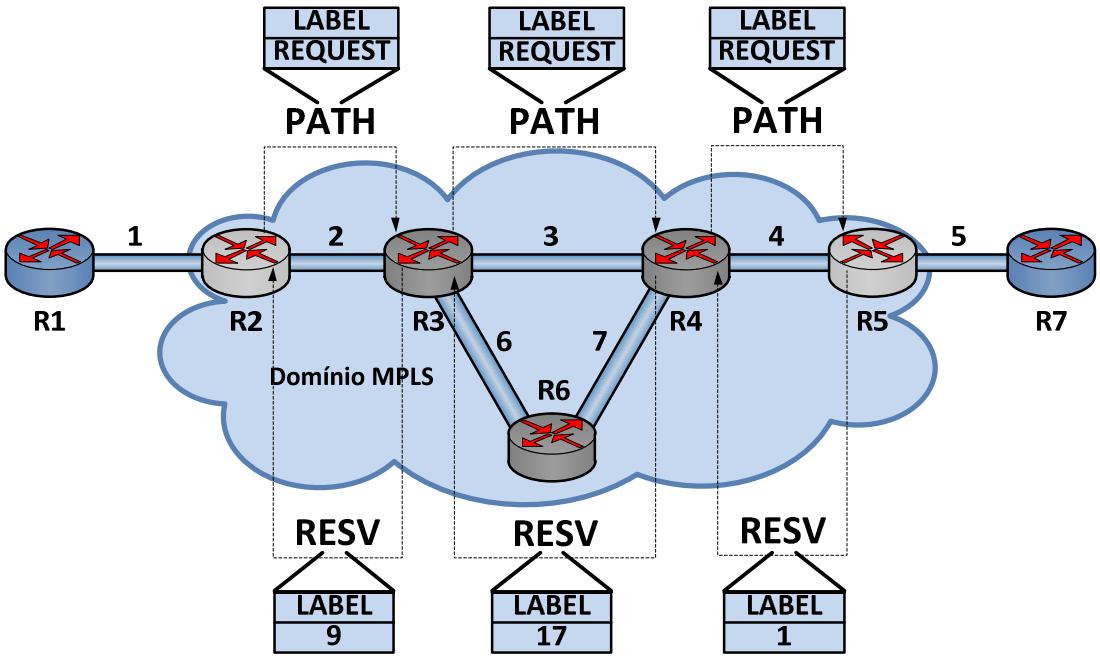


Fonte: Martins *et al* (2003)

As mensagens do tipo PATH solicitam ao próximo roteador do caminho selecionado, um Rótulo (*Label*) para o novo LSP a ser estabelecido, e os roteadores de egresso respondem em suas mensagens do tipo RESV os seus rótulos reservados. Estes rótulos recebidos nas mensagens RESV alimentam as tabelas de informação de encaminhamento por rótulos (LIB) que serão utilizadas pelo plano de encaminhamento para encaminhar os pacotes referentes ao novo LSP criado.

A (figura 21) demonstra a troca de mensagens PATH com o objeto de solicitação de rótulo (*Label Request Object*), as respostas RESV com o objeto rótulo (*Label Object*) e as tabelas de informação de encaminhamento por rótulos (LIB) dos roteadores no caminho do novo LSP resultantes.

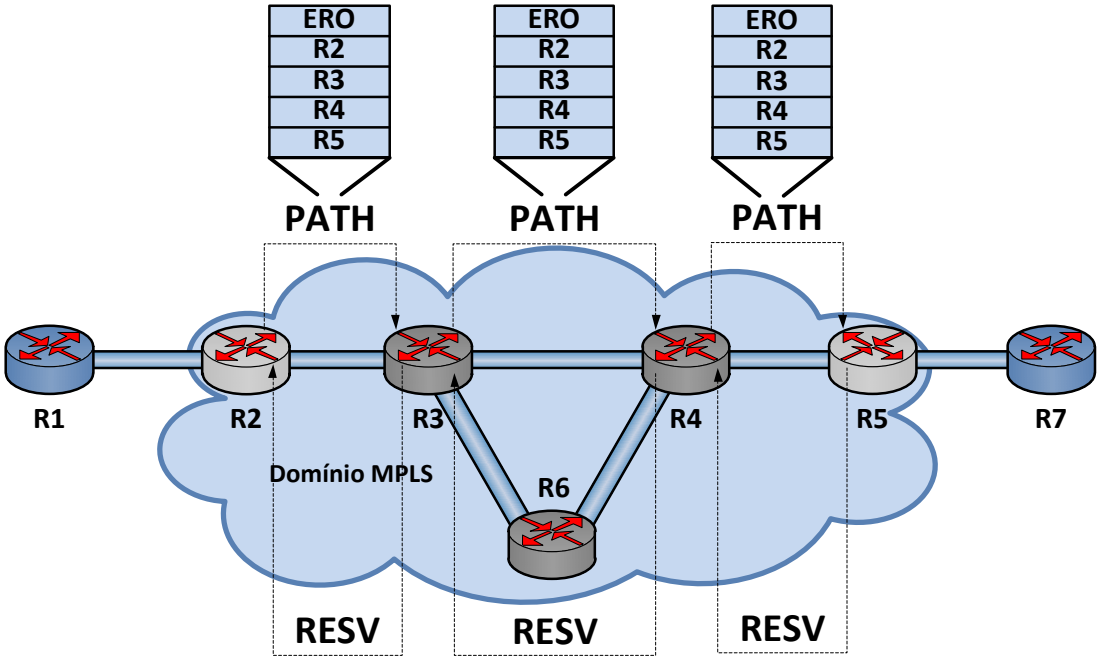
Figura 21 – Atualização da LIB pelo Protocolo RSVP-TE



Estado das LIBs - Após Estabelecimento			
Roteador 2 - Label Edge Router (LER)			
Int. de Entrada	FEC	Int. de Saída	Rót. de Saída
1	1	2	9
Roteador 3 - Label Switch Router (LSR)			
Int. de Entrada	Rót. de Entrada	Int. de Saída	Rót. de Saída
2	9	3	17
Roteador 4 - Label Switch Router (LSR)			
Int. de Entrada	Rót. de Entrada	Int. de Saída	Rót. de Saída
3	17	4	1
Roteador 5 - Label Edge Router (LER)			
Int. de Entrada	Rót. de Entrada	Int. de Saída	Rót. de Saída
4	1	5	-

A fim de possibilitar que o algoritmo de seleção de caminhos informe explicitamente o caminho que será utilizado para o estabelecimento de um novo LSP, na extensão do protocolo RSVP-TE foi criado o objeto de rota explícita (*Explicit Route Object* – ERO). O ERO contém o endereço de todos os roteadores no caminho do novo LSP que foi calculado pelo algoritmo de seleção de caminhos. Desta forma, os roteadores do caminho sabem quais são os próximos roteadores ao qual devem enviar suas mensagens PATH. O ERO é enviado na mesma mensagem PATH que contém o objeto de solicitação de rótulo (*Label Request Object*) como pode ser observado na figura 22.

Figura 22 – Objeto de Rota Explícita do Protocolo RSVP-TE



2.8 DS-TE

A arquitetura DS-TE (*DiffServ Aware MPLS Traffic Engineering*) foi proposta por Le Faucher e Lai (2003) como forma de prover QoS e engenharia de tráfego em uma granularidade de classes. As redes DS-TE podem ser consideradas uma evolução das redes MPLS-TE, já que conservam as características globais de encaminhamento e controle das mesmas, vistas na seção anterior, além de introduzirem extensões para suportar o conceito de alocação de banda, seleção de caminhos baseado em restrições e estabelecimento de caminho, considerando-se múltiplas classes de tráfego (LE FAUCHER, 2005b).

Nas redes DS-TE é possível, mapear os fluxos (LSPs) com base em seus requisitos de QoS em classes de tráfego (CT), alocando os recursos necessários fim-a-fim. O mapeamento destes fluxos é realizado no plano de controle pelo algoritmo de seleção de caminhos baseado em restrições. Este, por estar em execução no plano de controle, pode ter acesso à base de dados de Engenharia de Tráfego (TED – *Traffic Engineering Database*) para escolher o melhor caminho não só para o novo LSP, mas para a rede como um todo, melhorando assim o número de LSPs que poderão ser atendidos na rede.

O principal diferencial das redes DS-TE para as redes MPLS-TE é a possibilidade de agrupar e gerir a qualidade de serviço em Classes de Tráfego. Le Faucher e Lai (2003) conceituam Classe de Tráfego como:

> O conjunto de troncos de tráfego que passam por um determinado enlace, que está governado por um conjunto específico de restrições de utilização de banda nos enlaces da rede. Uma Classe de Tráfego (CT) é utilizada para os propósitos de alocação da banda, para o roteamento baseado em restrições e para o controle de admissão nas redes DS-TE. Um determinado Tronco de Tráfego pertence ao mesmo CT em todos os enlaces da rede DS-TE

O conceito de Classe de Tráfego implica em um conjunto de restrições de utilização de banda (BCs – *Bandwidth Constraints*) para cada CT. Estas restrições, bem como o uso atual das larguras de banda por CT, são, nas redes DS-TE, informações essenciais que devem ser entendida por todos os componentes que antes operavam apenas com a visão do uso do enlace.

Le Faucher e Lai (2003) recomendam o uso de no máximo 8 (oito) classes de tráfego (CTs): da CT0 a CT7. Quanto maior o valor numérico da CT, maior é a sua prioridade. Desta forma, a CT0 é a classe de menor prioridade, enquanto a CT7 é a de maior prioridade. São necessárias pelo menos duas CTs para ser provida a diferenciação de serviços por classes de tráfego.

As redes DS-TE suportam a configuração de prioridades para casos de efetiva concorrência por recursos da rede. As prioridades em redes DS-TE são divididas em prioridade de estabelecimento (*setup*) e prioridade de manutenção (*hold*). Se um novo LSPy possuir maior prioridade de estabelecimento do que a prioridade de manutenção do LSPx já estabelecido, este poderá sofrer preempção para o estabelecimento do LSPy. As prioridades de estabelecimento e manutenção podem variar de 0 (zero) a 7 (sete), sendo 0 (zero) o valor mais prioritário e o valor 7 (sete) o valor menos prioritário.

O processo de encerramento de LSPs, devido à necessidade de recursos para estabelecer um novo LSP, é chamado de preempção. Estes são utilizadas em situações de competição de recursos em que é necessário garantir prioridade entre dois ou mais LSPs. São indispensáveis cuidados especiais com este processo, pois pode induzir o LSP que sofre preempção a tentar ser restabelecido, o que pode levar à preempção de outro LSP e este a um outro em um processo em cascata.

A arquitetura DS-TE introduziu a definição de TE-Classes. Uma TE-Classe é um par ordenado composto pela CT e prioridade de preempção. Segundo a definição, um domínio DS-TE pode ter uma CT com uma ou mais TE-Classes com prioridade diferentes e uma ou

mais TE-Classes com a mesma prioridade em CTs distintas. Como é possível uma combinação de 8 (oito) classes de tráfego (CT) com 8 (oito) prioridades, totalizam-se em 64 (sessenta e quatro) as possibilidades de TE-Classes. Le Faucher e Lai (2003) e Minei e Lucek (2005) sugerem a escolha de, no máximo, 8(oito) TE-Classes. As combinações possíveis possibilitam ao gerente de rede inúmeras configurações, introduzindo flexibilidade na aplicação de diferentes formas de prioridade/preempção.

Em geral pode-se inferir os seguintes resultados operacionais das TE-Classes:

- Na mesma CT
  - Prioridade igual – não permite preempção entre LSPs da mesma CT.
  - Diferentes prioridades – permitem preempção entre LSPs da mesma CT
- Em CT distintas
  - Prioridades distintas – permitem preempção entre LSPs de CTs distintas.
  - Prioridade igual – não permite preempção entre CTs distintas.

O processo de seleção de caminhos nas redes DS-TE é análogo ao processo de seleção de caminhos das redes MPLS-TE com a necessidade de extensões na base de dados de Engenharia de Tráfego (TED – *Traffic Engineering Database*), protocolo de seleção de caminhos baseado em restrições e divulgação dos estados dos enlaces. Estes agora necessitam informações de uso e limites de banda na granularidade de classes de tráfego, ao invés de somente por enlace como era feito nas redes MPLS-TE. Neste trabalho, indicamos o uso da extensão do CSPF (*Constrained Short Path First*) sugerida na RFC 4124 de forma que a banda disponível não seja calculada apenas por enlace, mas que seja calculada por CT (LE FAUCHER, 2005b).

Após a seleção[4], o estabelecimento do caminho é feito com o protocolo RSVP-TE, o qual é similar ao processo de estabelecimento do caminho na arquitetura MPLS-TE com a necessidade de estender o RSVP-TE para considerar a reserva de banda por CT[5] (AWDUCHE *et al*, 2001).

Uma das premissas das redes DS-TE é a adoção de um modelo de alocação de banda para definir as regras de que serão utilizadas para reserva de banda (BCs - *Bandwidth Constraints*) pelas CTs nos enlaces. O desempenho da rede está diretamente associado à escolha correta do

---

[4] Pelo protocolo de seleção de caminhos baseado em restrições escolhido.

[5] Devido a esta necessidade, foi adicionado um novo objeto à mensagem PATH chamado Class Type Object que contém a CT da nova LSP.

modelo de alocação de banda e do algoritmo de seleção de caminhos baseado em restrições adequado aos fluxos existentes nesta.

2.8.1 Modelos de Alocação de Banda

Para efetivarmos as regras e limites para utilização dos enlaces pelas classes de tráfego (CTs) em uma rede DS-TE, necessitamos da adoção de um modelo de alocação de banda. O modelo de alocação de banda define se um LSP será aceito, bloqueado ou sofrerá preempção em um determinado enlace.

O modelo de alocação de banda está associado a um algoritmo de seleção de caminhos que definem os enlaces utilizados por um novo LSP (rota/caminho) sendo criado. Uma escolha adequada do modelo de alocação de banda pode implicar diretamente na melhoria do desempenho de operação da rede como um todo.

Existem diversos modelos de alocação de banda, sendo os primeiros definidos em RFC no contexto de redes DS-TE: o *Maximum Allocation Model* (MAM) e o *Russian Doll Model* (RDM), que passaremos agora a descrever.

2.8.1.1 *Maximum Allocation Model* (MAM)

O modelo de alocação de banda MAM corresponde a uma versão mais básica entre os modelos de alocação de banda existentes e tem como objetivo principal a reserva de um valor máximo de banda para cada Classe de Tráfego (CT), através do mapeamento de uma restrição de banda (BC) para as classes de tráfego (CTs) (LE FAUCHEUR; LAI, 2005).

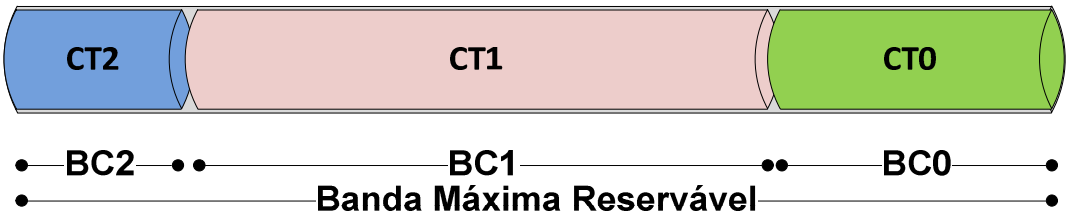
Adami e outros (2008) detalham mais especificamente o modelo MAM como segue:

- Para cada Classe de Tráfego "CTi" onde "M" é a banda máxima reservável no enlace e "N_i" é a banda alocada por CT tem-se:
  $$Ni \leq BCi \leq M$$

- Com as restrições,
  O total de banda alocada pelas CTs não pode exceder a capacidade do enlace $(\sum_{i=0}^{C-1} Ni \leq M)$.

Assim sendo, o somatório do total de banda alocada ocupado pelos LSPs (Ni) de uma determinada CT deve ser sempre menor ou igual ao valor máximo de restrição de banda (BC) associado a esta CT para um determinado enlace. Além disso, o somatório das alocações de banda das CTs corresponde sempre à banda disponível para alocação no enlace considerado (figura 23) (ADAMI et al, 2007).

Figura 23 – Modelo de Alocação de Banda MAM



A vantagem mais inerente do Modelo MAM é o isolamento total entre os tráfegos de CTs diferentes e, neste contexto, não se faz uso de mecanismos de preempção (ADAMI *et al*, 2007).

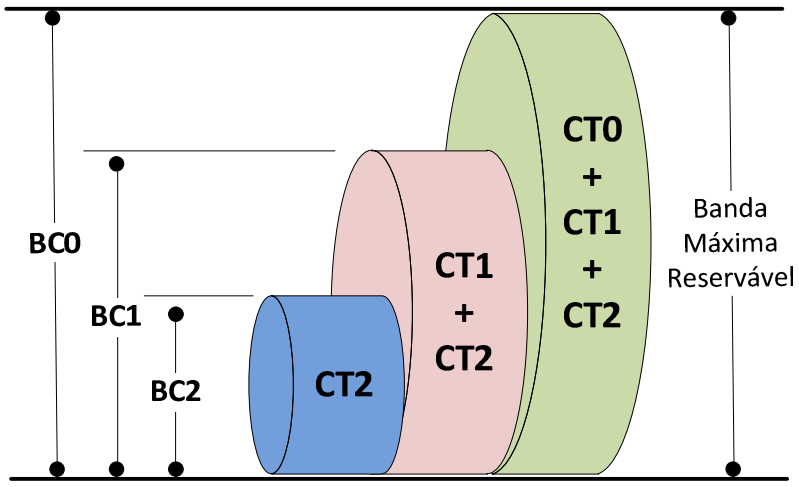
Um aspecto restritivo do modelo MAM no que diz respeito à utilização mais eficiente dos recursos de banda associados aos enlaces é não permitir o compartilhamento da banda não utilizada entre CTs. Em resumo, esta característica operacional implica na redução da eficiência em termos da utilização da banda máxima disponível por enlace (LE FAUCHER; LAI, 2005).

Para prover uma maior eficiência na utilização da banda, uma variação do MAM proposta por Tong e Yang (2007) permite que a soma dos BCs exceda a capacidade do enlace $(\sum_{i=0}^{C-1} BCi \geq M)$.

2.8.1.2 *Russian Doll Model* (RDM)

O diferencial do modelo RDM com relação ao modelo MAM está principalmente no compartilhamento da banda não utilizada, alocada para um CT de hierarquia superior por CTs hierarquicamente inferiores. No modelo RDM (figura 24) as CTs com maior valor numérico são hierarquicamente superiores em relação às CTs com menor valor numérico. O raciocínio geral envolvido na operação do modelo RDM é derivado do modelo básico de prioridades utilizado por aplicações em rede. No caso, a banda livre das aplicações mais prioritárias pode ser temporariamente utilizada por aplicações menos prioritárias. Neste cenário, considera-se a "preempção" de aplicações menos prioritárias por parte das aplicações prioritárias sempre que existir uma necessidade de banda por parte das aplicações prioritárias (LE FAUCHER, 2005a).

Figura 24 – Modelo de Alocação de Banda RDM



O modelo de compartilhamento de banda RDM apresenta como característica básica ser mais eficiente quanto à otimização da banda total utilizada se comparado com o modelo MAM. Uma consideração com relação ao modelo RDM é que este modelo não leva em conta a disponibilidade de banda em classes menos prioritárias e, também, o uso de preempções leva em princípio a uma inibição de serviço para as classes de aplicações mais prioritárias (LAI, 2005).

A tabela a seguir mostra as restrições de bandas no modelo MAM e RDM para as oito classes de tráfego configuradas na rede.

Tabela 5 - Restrições de Banda Modelo MAM X RDM para 8 CTs Configuradas em Rede

Restrição de Banda (BC)	Classe de Tráfego (CT) (MAM)	Classe de Tráfego (CT) (RDM)
BC7	CT7	CT7
BC6	CT6	CT7+CT6
BC5	CT5	CT7+CT6+CT5
BC4	CT4	CT7+CT6+CT5+CT4
BC3	CT3	CT7+CT6+CT5+CT4+CT3
BC2	CT2	CT7+CT6+CT5+CT4+CT3+CT2
BC1	CT1	CT7+CT6+CT5+CT4+CT3+CT2+CT1
BC0	CT0	CT7+CT6+CT5+CT4+CT3+CT2+CT1+CT0

Na tabela 6, os modelos MAM e RDM são comparados em relação às classes de tráfego (CTs) por restrições de banda (BCs), representação do valor máximo do BC, necessidade de preempção, eficiência de banda e proteção contra degradação (OSBORNE; SIMBA, 2003).

Tabela 6 – Comparativo entre os Modelos MAM X RDM

MAM	RDM
1 BC por CT	1 ou mais CT por BC
A soma de todas BCs pode exceder a capacidade do enlace.	BC0 é igual à capacidade máxima do enlace
Não é preciso preempção	É necessário preempções para prover garantia de banda para as CTs.
Eficiência da banda e proteção contra a degradação de QoS são mutuamente exclusivas.	Proporciona uma eficiência da banda e proteção contra a degradação QoS simultaneamente.

2.8.1.3 ADAPT-RDM

Pinto Neto e Martins (2008b) propuseram uma extensão do algoritmo RDM para redes DS-TE denominado ADAPT-RDM. O ADPT-RDM permite a definição de um fator de prioridade para ser utilizado em situações de preempção quando o limite da BC é excedido. Os autores simularam três fatores: preemptar LSP com menor prioridade, preemptar LSP com maior banda alocada e minimizar o número de LSPs preemptadas. O algoritmo é baseado em um gerenciamento centralizado conhecendo os LSPs configurados e estabelecidos na rede. O algoritmo também inclui o parâmetro opcional de frequência de preempção (*rate reduction*) para adequar a diferentes tamanhos de redes. O parâmetro anterior minimiza o re-roteamento e o tráfego de sinalização (PINTO NETO; MARTINS; BRITO, 2008).

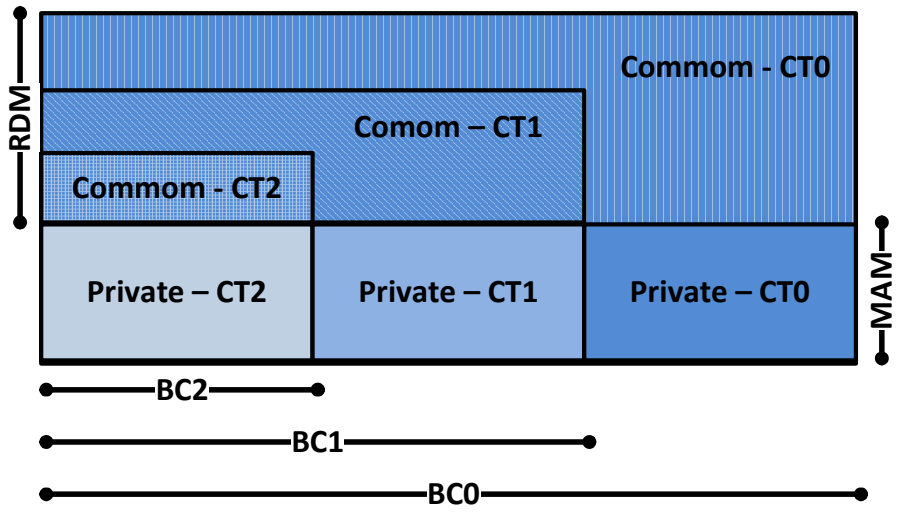
2.8.1.4 G-RDM

O Generalized-RDM (G-RDM), proposto por Adami e outros (2007), é uma solução híbrida do MAM e do RDM, que propõe reunir o melhor dos dois modelos de alocação de banda. Os autores criaram o conceito de banda compartilhada e banda privada das CTs. Os *private pools* formam a banda reservada da classe em que outras CTs não podem utilizar. *Common pools* são a banda compartilhada com outras CTs (ADAMI *et al*, 2007; ADAMI *et al,* 2010).

Como pode ser observado na

figura 25, o G-RDM divide virtualmente o enlace em dois. Na primeira partição, é aplicado o modelo MAM obtendo o efeito operacional de bandas privadas para as CTs com isolamento total entre as mesmas. Na segunda partição, é aplicado o modelo RDM com efeito operacional de compartilhamento de banda disponível de CTs de alta prioridade para CTs de baixa prioridade. Desta forma, o gerente da rede pode configurar reserva de bandas privadas e compartilhadas para as CTs.

Figura 25 - G-RDM (Generalized-RDM)



No modelo proposto no Capítulo 4, atinge-se uma melhor utilização dos enlaces e, em comparação ao modelo RDM, além de reduzir, também, o impacto da estratégia de preempções adotada pelo modelo RDM.

3 ALLOCCT-SHARING – UM MODELO PARA ALOCAÇÃO DE BANDA OPORTUNÍSTICA EM REDES DS-TE

Em uma rede DS-TE, os modelos de alocação de banda definem as regras e limites para utilização dos enlaces pelas classes de tráfego (CTs). Estas regras e limites refletem se um LSP será aceito, bloqueado ou sofrerá preempção em um determinado enlace.

Os modelos de alocação de banda existentes não consideram a situação na qual a banda disponível para classes de tráfegos inferiores possa vir a ser utilizada por classes de tráfegos superiores para melhoria de seus serviços. No conjunto atual de modelos de alocação de banda disponíveis na literatura e no limite do nosso conhecimento (Seção 3.3.1), as aplicações associadas à classe de tráfego mais prioritárias não utilizam banda de classes de tráfego inferiores, mesmo havendo disponibilidade de banda no enlace (banda não utilizada).

Com o objetivo de permitir uma melhor utilização dos enlaces através de um maior compartilhamento da banda, propõe-se o modelo de alocação de banda AllocCT-Sharing. Este modelo tem como alvo principal as redes multisserviço, nas quais aplicações e serviços prioritários e menos prioritários alocados em classes de tráfego disputam a utilização de banda como recurso de rede.

O AllocCT-Sharing é um modelo de alocação de banda com compartilhamento oportunista entre diferentes classes de tráfego e que tem como alvo redes, como as multisserviço, nas quais aplicações com prioridades distintas disputam por recursos do tipo “banda”.

O caráter oportunista do modelo de alocação de banda AllocCT-Sharing está em considerar a banda disponível em todas as CTs através do uso integrado de uma estratégia de compartilhamento de banda para CTs de prioridades inferiores por CTs de prioridades superiores (maior valor numérico) e, concomitantemente, estratégia de compartilhamento de banda para CTs de prioridades superiores por CTs de prioridades inferiores em um único modelo de alocação de banda.

O AllocCT-Sharing inova em relação aos modelos de alocação de banda existentes, compartilhando a banda disponível entre todas as classes de tráfego (CT), e com isso espera-se obter uma maior utilização do enlace.

Para compartilhar os recursos disponíveis de CTs de prioridades superiores para CTs de prioridades inferiores (menor valor numérico), o AllocCT-Sharing preserva a estratégia do algoritmo RDM. Para compartilhar os recursos disponíveis de CTs de prioridades inferiores para CTs de prioridades superiores (maior valor numérico), o AllocCT-Sharing permite uso temporário da banda de LSPs menos prioritários por parte dos LSPs mais prioritários

(empréstimos). Ocorre a efetiva "devolução" da banda sempre que existir uma necessidade por parte de um LSP (menos prioritário) do uso da banda alocada cedido por sua CT de menor prioridade para a CT de maior prioridade. Também é possível que a devolução dos empréstimos possa levar ao encerramento de um LSP mais prioritário estabelecido com uso da banda obtida por empréstimo (REALE; PINTO NETO; MARTINS, 2010).

Como resultado, o AllocCT-Sharing aceita um novo LSP sempre que houver banda disponível suficiente no enlace, seja esta banda de sua CT, de CTs hierarquicamente superiores ou inferiores, e aplica intervenções através de "preempção" e "devolução" apenas em situação de efetiva concorrência por recursos de banda na rede.

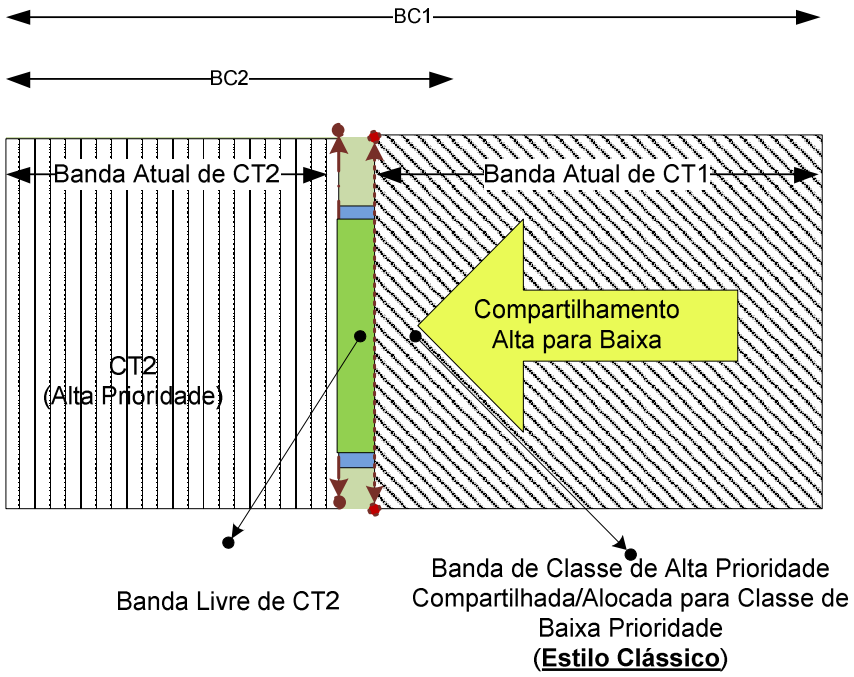
O AllocCT-Sharing define assim um novo modelo de compartilhamento para redes DS-TE, com a utilização do modelo RDM acrescido de empréstimos e devoluções.

3.1 ESTRATÉGIA DE COMPARTILHAMENTO "ALTA PARA BAIXA" E "BAIXA PARA ALTA"

O modelo básico de prioridades tem como estratégia permitir que a banda livre das aplicações mais prioritárias possa ser temporariamente utilizada por aplicações menos prioritárias. Como visto anteriormente, nas redes DS-TE, as aplicações são mapeadas em LSPs (*Label Switched Paths*) que, por sua vez, são agrupados em classes de tráfego (CT).

O modelo de alocação de banda RDM e seus sucessores aplicam a estratégia do modelo básico de prioridades, como pode ser observado na (figura 26), ao permitir o compartilhamento de banda de CTs de prioridades superiores (maior valor numérico) para LSPs de CTs de prioridades inferiores. Cabe enfatizar que, nesta estratégia, utiliza-se o mecanismo de "preempção" sempre que existir uma necessidade de banda por parte de LSPs prioritários que leve à desalocação de recursos de um LSP menos prioritário.

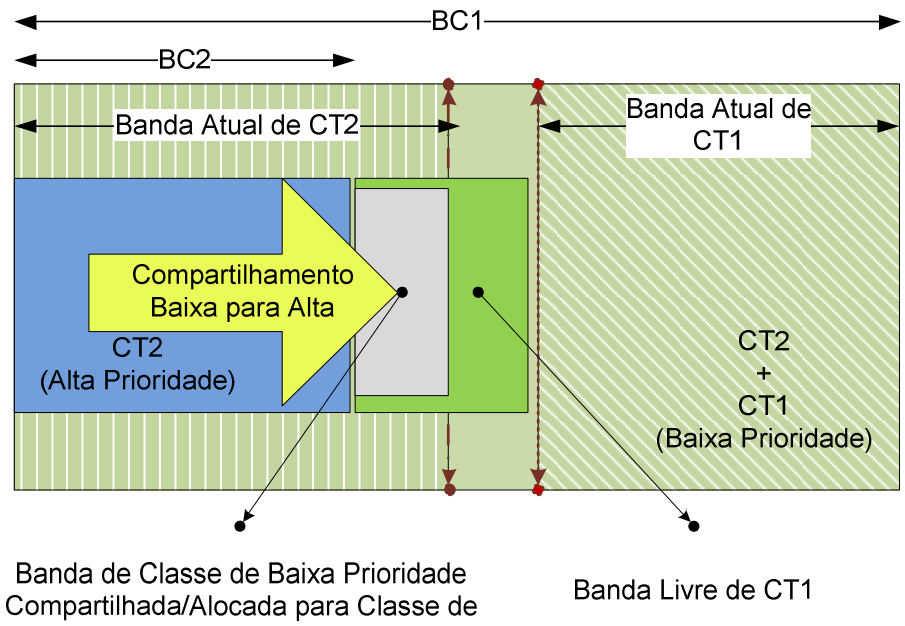
Figura 26 - Compartilhamento “Alta para Baixa”



Os modelos de alocação de banda derivados do modelo básico de prioridades não consideram que a banda livre de aplicações menos prioritárias possa ser temporariamente utilizada por aplicações mais prioritárias.

Com o intuito de permitir o compartilhamento de banda para CTs de prioridades superiores por CTs de prioridades inferiores novos conceitos foram definidos conforme a (figura 27).

Figura 27 – Compartilhamento “Baixa para Alta”



Classificamos como estratégias de compartilhamento “baixa para alta” (*low-to-high*) as estratégias de compartilhamento de banda de CTs de prioridades inferiores para CTs de prioridades superiores (maior valor numérico); e estratégias de compartilhamento “alta para baixa” (*high-to-low*) as estratégias de compartilhamento de banda de CTs de prioridades superiores para CTs de prioridades inferiores.

Para a operacionalização da estratégia de compartilhamento “baixa para alta” introduzimos os conceitos de “**Empréstimo**” e “**Devolução**”.

O conceito de “**Empréstimo”** corresponde ao uso temporário de banda de LSPs menos prioritários por parte dos LSPs mais prioritários.

O “empréstimo” pode ser temporário segundo as regras definidas adiante pelo modelo de alocação de banda. O conceito de “preempção” não pode ser aplicado no caso de devolução do “empréstimo”, por não se tratar de uma relação de LSPs prioritários que tomam recursos LSPs menos prioritários: seria justamente o inverso.

Define-se, então, o termo “**devolução**”. Uma “**devolução**” ocorre sempre que existir uma necessidade de banda por parte de um LSP menos prioritário que leve à devolução dos recursos (banda) de um LSP mais prioritário **estabelecido por empréstimo**.

Diante dos conceitos e classificações definidos anteriormente, podemos apresentar o modelo AllocCT-Sharing com sua proposta de integração das estratégias de compartilhamento “baixa para alta” e “alta para baixa” em um único modelo de alocação de banda.

3.2 O MODELO ALLOCCT-SHARING

O modelo AllocCT-Sharing é o resultado da integração das duas estratégias de compartilhamento de banda apresentadas anteriormente, “alta para baixa” e “baixa para alta”, em um único modelo de alocação de banda, com o objetivo de permitir uma melhor utilização da banda dos enlaces da rede através de um maior compartilhamento entre as CTs. No modelo, considera-se que as classes de tráfego (CTs) inferiores podem utilizar a banda não utilizada de CTs hierarquicamente superiores e vice-versa.

Como resultado desta estratégia mais flexível, o AllocCT-Sharing aceita um novo LSP sempre que houver banda disponível suficiente no enlace seja esta banda de sua CT, de CTs hierarquicamente superiores ou inferiores e aplica intervenções do tipo “preempção” e “devolução” apenas em situações de efetiva concorrência por recursos de banda no enlace.

A partir de um maior compartilhamento entre as CT espera-se que o número de LSPs bloqueados e com ação de preempção seja reduzido, já que as ações de bloqueio e preempção ocorrem apenas em casos de saturação do enlace (uso de toda a banda do enlace). Também se espera que nem todo LSP estabelecido que solicite banda emprestada de outro CT precise devolvê-la, pois, possivelmente, o LSP pode ser encerrado antes de ser solicitada a devolução do empréstimo pela CT proprietária.

Nos cenários onde existe a saturação do enlace, o AllocCT-Sharing converge para as restrições de banda (BC) configuradas, de forma a preservar as garantias mínimas de banda previstas nos acordos de serviço (SLAs), pois a operação do modelo reflete a devolução dos empréstimos como situação primária. Após a devolução dos empréstimos, o algoritmo opera num cenário de compartilhamento “alta para baixa”.

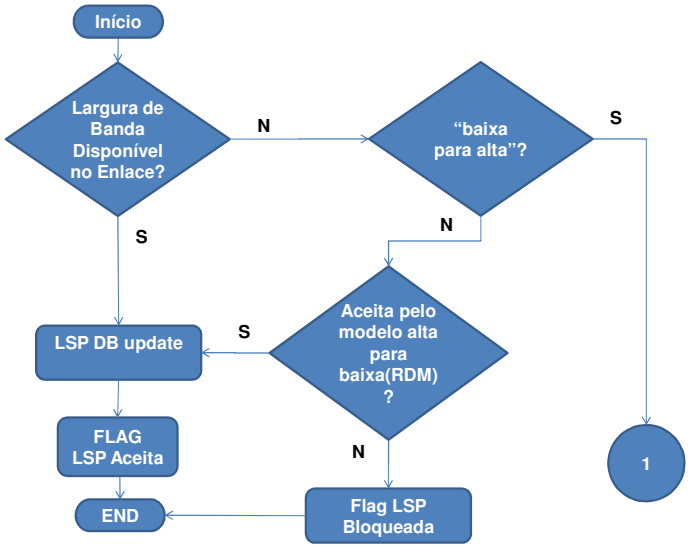
A operação do AllocCT-Sharing preserva a abordagem do algoritmo RDM e seus sucessores para alocação de banda "alta para baixa" e introduz uma nova abordagem de alocação de banda com "empréstimos" e “devoluções” para alocação de banda "baixa para alta", a fim de procurar alcançar um melhor compartilhamento de banda entre aplicações na rede.

O funcionamento global do modelo pode ser observado nos fluxogramas ilustrados nas figuras 28 e 30 que passaremos agora a detalhar.

Como característica inerente do novo modelo, enquanto houver recursos de banda no enlace, todos os LSPs são aceitos seja por compartilhamento "baixa para alta", ou “alta para baixa”.

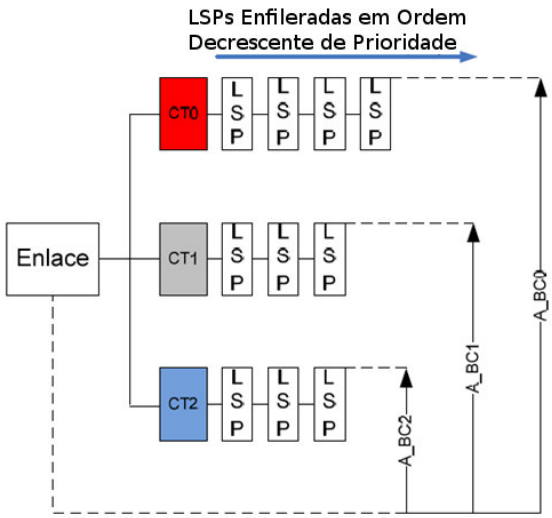
Devido a esta característica, ao chegar um pedido de estabelecimento de um novo LSP, a primeira ação do AllocCT-Sharing é verificar se existe banda disponível no enlace, como pode ser observado no primeiro teste condicional do fluxograma da figura 28.

Figura 28 – AllocCT-Sharing - Conflito “Alta para Baixa”



Após este teste, e caso exista banda disponível no enlace, as listas de LSP estabelecidos no enlace agrupados por CT são atualizadas na base da entidade gerenciadora, como pode ser observado na figura 29 e é sinalizado para o algoritmo de seleção de caminho que o LSP foi aceito naquele enlace.

Figura 29 - LSPs Agrupados por CT na Base de Dados da Entidade Gerenciadora



O efetivo estabelecimento do LSP (*path setup*) será feito a partir do caminho selecionado pelo algoritmo de seleção de caminhos utilizado, aceito pelo modelo AllocCT-Sharing, e efetivado pelo protocolo de estabelecimento de LSPs (*path setup*) como, por exemplo, o RSVP-TE (*Resource Reservation Protocol - Traffic Engineering*) com sua extensão para o DS-TE (*DiffServ Aware MPLS Traffic Engineering*).

Quando não houver mais banda disponível no enlace para estabelecer a nova solicitação de LSP, situação onde há efetiva disputa por recursos de banda no enlace (saturação do enlace), o AllocCT-Sharing precisa acessar a base de dados, anteriormente alimentada a cada chegada de novos LSPs, e verificar se é uma situação de compartilhamento "baixa para alta" ou “alta para baixa”.

Essa situação é representada pelo condicional “é baixa para alta?”, pois se não houver restrições de banda (BC) excedidas, conforme modelo “alta para baixa”, estamos diante de uma situação de compartilhamento típico “alta para baixa” e devemos aplicar uma restrição de banda para este cenário. Caso exista uma ou mais restrições de banda excedida, estamos diante de uma situação de compartilhamento “baixa para alta”.

Nas situações de compartilhamento “alta para baixa”, optou-se por aplicar o modelo RDM, pois se considera que essa estratégia apresenta resultados positivos, dispõe-se do código de implementação e simulação para a realização de comparações e validação; além disto, o RDM é a origem efetiva e eficiente das demais propostas com estratégias de compartilhamento de banda para CTs de prioridades inferiores por CTs de prioridades superiores (maior valor numérico) (ADAMI *et al*, 2007; PINTO NETO; MARTINS, 2008a; PINTO NETO; MARTINS, 2008b; PINTO NETO, 2008).

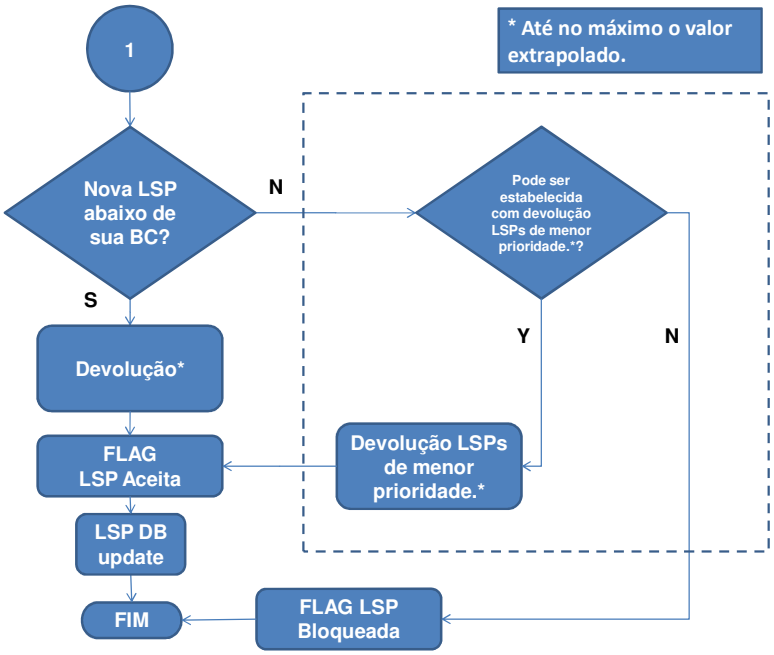
Enfatizamos que qualquer outro modelo de alocação de banda com estratégia de compartilhamento “alta para baixa” pode ser aplicado desde que retorne para o AllocCT-Sharing a lista de LSPs sujeitos à preempção ou se o novo LSP foi bloqueado.

O modelo AllocCT-Sharing precisa das informações de LSP estabelecidos nos enlace por CT e banda requisitada para o novo LSP para tomar as decisões de preempção, bloquear solicitação, devolução de empréstimo e outros, além de poder acionar o protocolo de estabelecimento e liberação de LSPs (RSVP-TE e outros).

Caso seja confirmada pelo AllocCT-Sharing uma situação de compartilhamento “alta para baixa” é chamado o algoritmo de compartilhamento “alta para baixa”, conforme descrição do parágrafo anterior. Por fim o AlloCT-Sharing sinaliza para o algoritmo de seleção de caminho se o novo LSP puder ser estabelecida ou não e atualiza a base de dados.

Outra situação a ser considerada é quando há a saturação do enlace com uma ou mais restrições de banda excedidas. Neste caso, há uma situação de compartilhamento “baixa para alta” com empréstimos de banda para CTs de prioridades superiores por CTs de prioridades inferiores. Neste cenário executa-se o algoritmo ilustrado na figura 30.

Figura 30 – AllocCT-Sharing - Conflito “Baixa para Alta”



Como pode ser observado no primeiro condicional da figura 30, o primeiro passo do AllocCT-Sharing é verificar se a CT do novo LSP está abaixo do valor configurado como sua restrição de banda (BC). Esta verificação tem como objetivo saber se a CT da novo LSP emprestou banda para outras CTs.

Em caso afirmativo, a CT do novo LSP emprestou banda para outra CT e irá solicitar a devolução para o estabelecimento de um LSP pertencente à sua classe (CT). Para isto, o AllocCT-Sharing procede a “devolução” com o auxílio, por exemplo, do protocolo RSVP-TE, que libera os recursos de um ou mais LSPs de forma a liberar a banda necessária para o estabelecimento do novo LSP.

Se as situações anteriores não forem possíveis, o AllocCT-Sharing informa o bloqueio do novo LSP ao algoritmo de seleção de caminhos.

É importante observar que na figura 30 pode ser visto um conjunto de operações representadas com uma caixa tracejada ao redor. Estas operações permitem aprimorar o

AllocCT-Sharing para verificar a possibilidade de aplicar prioridades aos empréstimos antes de bloquear o novo LSP.

Ao aplicarmos prioridades aos empréstimos verificamos, antes de bloquear o LSP, a possibilidade de "devolução" de empréstimos de LSPs com menor prioridade do que o novo LSP a ser estabelecido.

Caso a verificação implementada como aprimoramento seja positiva, o AllocCT-Sharing faz a devolução de empréstimos, da menor para a maior prioridade de um ou mais LSPs, para liberar a banda necessária. Para o estabelecimento do novo LSP, sinaliza para o algoritmo de seleção de caminho que o novo LSP pode ser estabelecido e atualiza as informações do enlace. O procedimento de devolução de LSP e estabelecimento são feitos com auxílio de um protocolo como o RSVP-TE.

Este aprimoramento foi implementado, a fim de reconhecer as prioridades dos LSP em conformidade com o DS-TE.

A tabela 7 apresenta as principais características dos modelos de alocação de banda MAM, RDM e AllocCT-Sharing.

Tabela 7 - Comparativo entre MAM, RDM e AllocCT-Sharing

	MAM	**RDM**	**AllocCT-Sharing**
Compartilhamento "alta para baixa"	Não	Sim	Sim
Compartilhamento "baixa para alta"	Não	Não	Sim
Eficiência na utilização da banda com alto tráfego de baixa prioridade	Baixa	Alta	Alta
Eficiência na utilização da banda com alto tráfego de alta prioridade	Baixa	Baixa	Alta
Isolamento entre CTs	Alta	Média	Baixa

Como pode ser observado na tabela 7, o modelo de alocação de banda MAM é indicado para cenários onde é desejado um alto isolamento entre CTs. O isolamento total entre as CTs, onde não há compartilhamento entre CTs, implica em uma baixa eficiência na utilização da banda do enlace, já que a banda disponível de uma CT não pode ser utilizada por outra CT.

Já o modelo RDM reduz o isolamento entre as CTs, permitindo que a banda disponível das CTs mais prioritárias possam vir a ser utilizadas pelas CTs menos prioritárias (modelo básico

de prioridades). Esta estratégia permite uma maior eficiência na utilização da banda dos enlaces em cenários onde o tráfego de alta prioridade é baixo e o tráfego de baixa prioridade é alto, mas conserva as restrições de compartilhamento no cenário oposto.

O modelo AllocCT-Sharing introduz um caráter oportunista, permitindo um compartilhamento total entre as CTs de forma a compartilhar toda e qualquer banda disponível no enlace, seja por compartilhamento "alta para baixa" implementado pelo modelo RDM ou por compartilhamento "baixa para alta" através de empréstimos e devoluções. Este compartilhamento permite uma alta eficiência na utilização do enlace tanto nos cenários onde o tráfego de alta prioridade é baixo e o tráfego de baixa prioridade é alto como no cenário oposto. Como poderá ser observado no estudo de caso abaixo.

3.2.1 Estudo de Caso Comparativo: AllocCT-Sharing versus RDM

Nesta seção, é ilustrado um exemplo numérico em que a restrição do compartilhamento de banda disponível de CTs de prioridade inferiores para LSPs de CTs de prioridades superiores ("baixa para alta") é verificada no modelo RDM e em seguida o exemplo numérico onde o compartilhamento "baixa para alta" é aceito pelo modelo AllocCT-Sharing.

O objetivo deste estudo de caso é demonstrar o caráter oportunista e flexível do modelo AllocCT-Sharing ao implementar as estratégias de compartilhamento "alta para baixa" e "baixa para alta" otimizando o uso da banda do enlace.

A tabela 9 mostra a banda e a CT correspondente para um conjunto de LSPs estabelecidos em dado momento para um determinado enlace hipotético de 200Mbps. O enlace está configurado com três CTs (CT0, CT1 e CT2), sendo CT2 de mais alta hierarquia e CT0 mais baixa hierarquia para o modelo RDM configurado.

Os valores de restrições de banda (BCs) configurados estão apresentados na tabela 8.

Tabela 8 - Restrições de Banda (BC) Configurada no Enlace

BC	Max BC (%)	MAX BC (Mbps)	CT por BC
BC0	100	200	CT0+CT1+CT2
BC1	70	140	CT1+CT2
BC2	40	80	CT2

Tabela 9 - Configuração dos LSPs Estabelecidos - RDM

Linha	LSP	CT	Bandwidth *	BC2 *	BC1 *	BC0 *	Disponível no Enlace *	Status
1	0	2	80	0	60	120	120	Aceita
2	1	1	40	0	20	80	80	Aceita
3	2	0	20	0	20	60	60	Aceita
4	**3**	**2**	**40**	**- 40**	**- 20**	**20**	**20**	**Bloqueada**
5	**4**	**1**	**25**	**0**	**-5**	**35**	**35**	**Bloqueada**

Nota: * Em Mbps

Em uma situação hipotética, prevê-se, por exemplo, que uma requisição de estabelecimento de um novo LSP (LSP3 com banda = 40Mbps) chegue associado à CT2 (conforme linha 4 – taela 9). De acordo com as restrições do modelo RDM, o mesmo não poderá ser estabelecido e será bloqueado. Este fato é decorrente da CT2 já ter utilizado os 80Mbps configurados para BC2 e a aceitação do novo LSP implicaria em extrapolar em 40 Mbps o valor da BC2.

O mesmo ocorreria para o pedido de estabelecimento de um novo LSP (LSP4 com banda = 25Mbps) associado à CT1 que tem como restrição de banda (BC1) (configurada com 140Mbps linha 5 – tabela 9. A CT1 já utiliza 120Mbps (80Mbps da CT2 + 40Mbps da CT1) e com a nova solicitação extrapolará em 5Mbps a sua restrição BC1.

Pode ser observado ainda que nos dois casos exista banda disponível em CTs inferiores à CT do novo LSP, mas esta banda não pode ser utilizada devido às restrições impostas pelo modelo RDM (“alta para baixa”), usado até então e base dos diferentes modelos de alocação de banda que evoluíram a partir dele (ADAMI *et al*, 2007; PINTO NETO; MARTINS, 2008a; PINTO NETO; MARTINS, 2008b; PINTO NETO, 2008).

Aplicando o modelo de alocação de banda AllocCT-Sharing, ao mesmo cenário proposto anteriormente, as chegadas dos LSP 3 e LSP 4, não se provoca a saturação do enlace, e conforme o fluxograma apresentado anteriormente, os LSP são admitidos através do compartilhamento “alta para baixa” implementado pelo algoritmo.

Podemos observar nas linhas 4 e 5 da tabela 10 o estado da rede com a aceitação dos dois LSPs.

Tabela 10 - Configuração dos LSPs Estabelecidos – AllocCT-Sharing - Empréstimos

Linha	LSP	CT	Bandwidth *	BC2 *	BC1 *	BC0 *	Disponível no Enlace *	Status
1	0	2	80	0	60	120	120	Aceita
2	1	1	40	0	20	80	80	Aceita
3	2	0	20	0	20	60	60	Aceita
4	**3**	**2**	**35**	**- 35**	**- 15**	**25**	**25**	**Aceita**
5	**4**	**1**	**25**	**-35**	**-40**	**0**	**0**	**Aceita**

Nota: * Em Mbps

A implementação do compartilhamento “baixa para alta” pelo modelo AllocCT-Sharing faz uso do novo conceito de “empréstimo” que corresponde ao uso temporário de banda de LSPs menos prioritários por parte dos LSPs mais prioritários. Como os empréstimos são ocasionalmente temporários e estão sujeitos à devolução, será computado na atualização da base de dados de controle do algoritmo AllocCT-Sharing que:

- a CT2 utilizou 35 Mbps em empréstimo das CTs inferiores para estabelecer o LSP 3, pois não tinha nenhuma banda disponível para sua CT.
- a CT1 utilizou 5 Mbps em empréstimo das CTs inferiores para estabelecer seu LSP 4.

O valor de empréstimo utilizado em CT1 é obtido através da subtração do valor extrapolado da restrição de banda da BC1 do valor extrapolado da restrição de banda da BC0.
Observa-se na tabela 11 que a solicitação de estabelecimento de um novo LSP (linha 6) com 40 Mbps associada à CT0 implicará em uma situação de efetiva concorrência por recursos da rede já que não há banda disponível no enlace.

Tabela 11 - Configuração dos LSPs Estabelecidos – AllocCT-Sharing – Devolução de Empréstimos

Linha	LSP	CT	Bandwidth *	BC2 *	BC1 *	BC0 *	Disponível no Enlace *	Status
1	0	2	80	0	60	120	120	Aceita
2	1	1	40	0	20	80	80	Aceita
3	2	0	20	0	20	60	60	Aceita
4	3	2	35	- 35	- 15	25	25	Devolução
5	4	1	25	-35	-40	0	0	Devolução
6	**5**	**0**	**40**	**0**	**0**	**20**	**20**	**Aceita**

Nota: * Em Mbps

Nesta situação, o modelo AllocCT-Sharing percebe que é uma situação de concorrência “baixa para alta”, pois existe uma ou mais restrições de banda (BC) acima do valor configurado. Diante das considerações anteriores, o AllocCT-Sharing faz a devolução da

banda alocada pelos LSP 3 e LSP 4 (linhas 4 e 5 - tabela 11) que foram estabelecidas por empréstimos. Observa-se que os LSP 3 e LSP 4 são pertencentes a CTs mais prioritárias, mas foram estabelecidas com banda "emprestada".

Após ter feito a devolução de banda à CT, o AllocCT-Sharing sinaliza o aceite da LSP5 para o algoritmo de seleção de caminhos (linha 6 - tabela 11).

O modelo de alocação de banda está associado a um algoritmo de seleção de caminhos para que este defina os enlaces utilizados por um novo LSP (rota/caminho) a ser criado. Este algoritmo de seleção de caminhos precisa estar em harmonia com o modelo de alocação de banda para uma operação adequada, e eventualmente, mais eficiente da alocação de banda na rede como um todo.

3.3 INTEGRAÇÃO DO ALGORITIMO DE ALOCAÇÃO DE BANDA (ALLOCCT-SHARING) COM O ALGORÍTIMO DE SELEÇÃO DE CAMINHO (CSPF)

Ao chegar uma nova requisição de LSP em rede habilitada com DS-TE, é necessário computar um caminho na rede (*path*), ou seja, uma sequência de enlaces por onde o novo LSP será estabelecido entre sua origem e seu destino. De maneira geral, esta é a tarefa dos algoritmos de seleção de caminhos como, por exemplo, o CSPF (*Constrained Short Path First*) (ENNE, 2009).

A escolha do caminho em uma rede habilitada com DS-TE precisa ir além de simplesmente otimizar uma métrica como número de saltos, banda disponível etc. É preciso também que esse algoritmo não viole um conjunto de restrições (BCs) em consonância com o modelo de alocação de banda utilizado. Os algoritmos de definição de caminho que buscam encontrar uma rota que otimize certa métrica e, ao mesmo tempo, não viole um conjunto de restrições, são denominados algoritmos de cálculos de caminho baseado em restrições.

Para o correto funcionamento do algoritmo de seleção de caminhos baseado em restrições e dos modelos de alocação de banda em redes DS-TE, é necessário que as bandas disponíveis nos enlaces sejam divulgadas tendo como base a informação das CTs. Como vimos anteriormente, no capítulo sobre DS-TE, as CTs (Classes de Tráfego) correspondem a um conjunto de LSPs (*traffic trunks*) agrupados segundo um mesmo critério gerencial e regidos por um modelo de restrição de banda (LAI, 2005; LE FAUCHER; LAI, 2005; LE FAUCHER, 2005(a); LE FAUCHER; LAI, 2003).

Assim sendo, a efetivação (*setup*) de um LSP implica em selecionar um caminho e considerar ao mesmo tempo as restrições de banda com um modelo como o AllocCT-Sharing. Em

seguida, indicamos nossa opção em termos de seleção de caminhos e sua integração com o modelo de alocação de banda AllocCT-Sharing.

Dentre os diversos algoritmos de seleção de caminho existentes na literatura, optamos por focar nosso trabalho na extensão do CSPF sugerida na RFC 4124, de forma que a banda disponível não seja definida apenas por enlace, mas por CT. A partir desta, extensão, o algoritmo CSPF pode calcular o menor caminho na rede em que exista banda disponível na CT por enlace, de acordo com as restrições de banda impostas pelo modelo de alocação de banda (LE FAUCHER, 2005b).

Na extensão sugerida na RFC 4124, cada caminho é testado por ordem crescente de número de saltos, e se existir banda correspondente para ser utilizada em todos os enlaces do caminho na CT associada à nova requisição, o mesmo é selecionado. Se em nenhum dos caminhos existentes houver banda disponível para a CT ao qual a nova requisição está associada, a mesma é bloqueada. Em resumo, o critério de seleção de caminho efetivamente utilizado é aquele de menor custo (*hops*) com restrição de banda considerada.

A extensão da RFC 4124 é necessária, pois o algoritmo de seleção de caminhos deve estar integrado ao modelo de alocação de banda para a melhor escolha dos enlaces (caminho) do novo LSP sendo solicitado.

Neste trabalho, adaptamos o algoritmo de seleção de caminho CSPF com a extensão sugerida na RFC 4124, para que o mesmo trabalhe de forma "oportunista" com o conhecimento dos empréstimos junto ao novo modelo AllocCT-Sharing proposto (LE FAUCHER, 2005b).

Chamamos de "*Loan CSPF*" a integração do modelo de alocação de banda AllocCT-Sharing com o algoritmo de seleção de caminhos CSPF.

3.3.1 Loan CSPF

Como visto na seção anterior, a proposta de integração do algoritmo de seleção de caminhos CSPF para trabalhar em conjunto com o modelo de alocação de banda AllocCT-Sharing partiu da extensão sugerida na RFC 4124 (LE FAUCHER, 2005b).

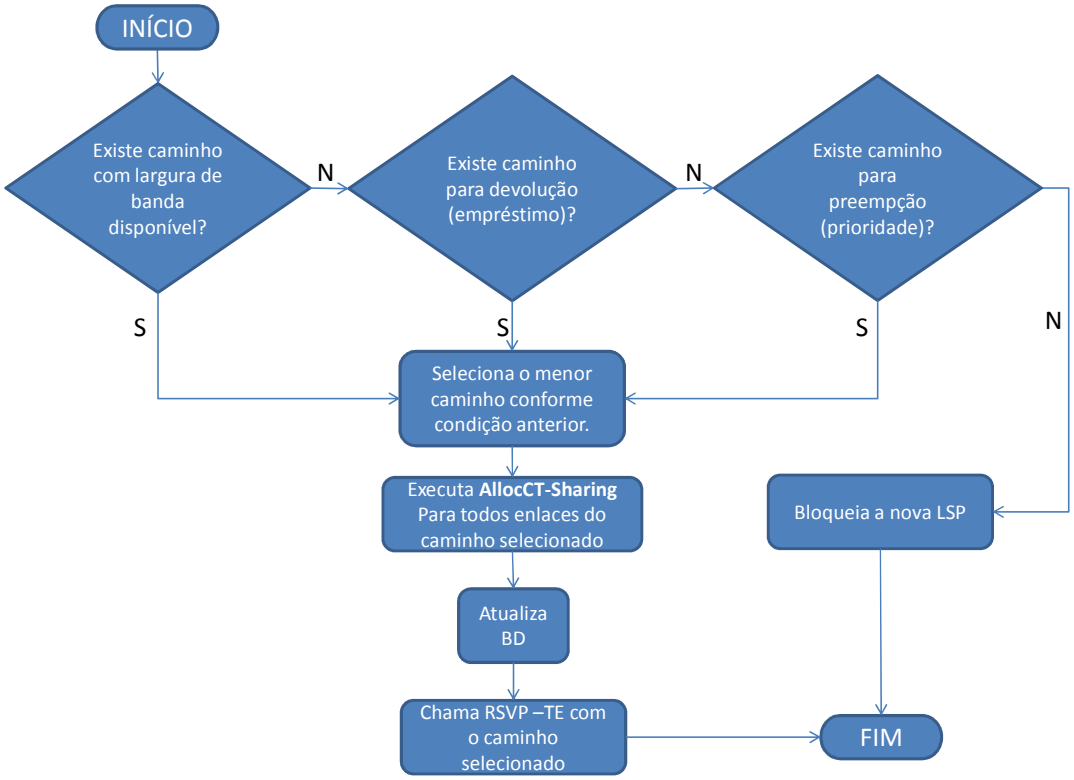
Segundo esta extensão, a banda disponível é calculada por CT e cada caminho é testado por ordem crescente de número de saltos. A solicitação de estabelecimento de um novo LSP é aceita, existindo banda na CT associada à nova requisição do LSP em todos os enlaces do caminho.

A integração do modelo de alocação de banda AllocCT-Sharing com o algoritmo de seleção de caminhos CSPF (*Loan CSPF*) considerou os seguintes aspectos da operação do modelo AllocCT-Sharing:

- Utilização simultânea de estratégias de compartilhamento “baixa para alta” e “alta para baixa”;
- Existência de “empréstimos” e
- Prioridade entre classes de tráfego.

A combinação da recomendação contida na RFC 4124 com as adaptações necessárias devido aos aspectos refletidos da operação do modelo de alocação de banda AllocCT-Sharing resulta no fluxograma representado na figura 31 que passaremos a descrever.

Figura 31 - Fluxograma do *Loan CSPF*



Segundo o fluxograma da figura 31, ao chegar uma solicitação de estabelecimento de LSP, verifica-se a existência de pelo menos um caminho com banda disponível. Em caso afirmativo, é selecionado o menor caminho com banda disponível e:

- Executa o AllocCT-Sharing para todos os enlaces do caminho selecionado. Este procedimento verifica se o novo LSP atende aos requisitos do AllocCT-Sharing (O sinal “LSP ACEITO” deve ser retornado para todos os enlaces do caminho selecionado)
- Atualiza a base de dados e

- Chama o RSVP-TE com o caminho selecionado para o efetivo estabelecimento do novo LSP.

Caso não exista nenhum caminho com banda disponível, estamos diante a uma efetiva concorrência por recursos da rede. Nesta situação existem duas possibilidades para liberar recursos de banda para que o novo LSP possa ser aceito:

- Devolução de LSPs. – Devoluções podem ser feitas sempre que existe uma necessidade de banda por parte de um LSP menos prioritária que leve ao encerramento de um LSP mais prioritário estabelecido por empréstimo (Empréstimo).
- Preempção de LSPs. – Preempções podem ser executadas sempre que existir uma necessidade de banda por parte de LSPs prioritários que leve ao encerramento de uma LSP menos prioritário, respeitando as restrições de bandas (BCs) configuradas (Prioridade).

Caso exista um ou mais caminhos com possibilidade de devolução de empréstimo para o estabelecimento do novo LSP, os seguintes passos são realizados:

- Executa o AllocCT-Sharing para todos os enlaces do menor caminho selecionado. Este procedimento faz a devolução dos empréstimos necessários para o estabelecimento do novo LSP (O sinal "LSP ACEITO" deve ser retornado para todos os enlaces do caminho selecionado) [6].
- Atualiza a base de dados e
- Chama o RSVP-TE com o caminho selecionado para o efetivo estabelecimento do novo LSP.

Caso as condições anteriores sejam negativas, o algoritmo de seleção de caminhos verifica a existência de um ou mais caminhos em que possa ser feita a preempção de LSP, encerramento de LSPs de menor prioridade que o novo LSP, e executa os seguintes passos:

- Executa o AllocCT-Sharing para todos os enlaces do menor caminho selecionado. Este procedimento faz a preempção de LSP menos prioritárias necessárias para o estabelecimento do novo LSP (O sinal "LSP ACEITO" deve ser retornado para todos os enlaces do caminho selecionado) [6].
- Atualiza a base de dados e

[6] AllocCT-Sharing utiliza o RSVP-TE para encerrar os LSP na devolução de empréstimos.

- Chama o RSVP-TE com o caminho selecionado para o efetivo estabelecimento do novo LSP.

Na falta de caminhos com banda suficiente, possibilidade de devoluções e possibilidade de preempção, o algoritmo de seleção de caminhos bloqueia o novo LSP.

Modelo AllocCT-Sharing – Simulação e Prova de Conceitos

A prova de conceito foi realizada, considerando uma avaliação comparativa da característica oportunista do modelo de alocação de banda AllocCT-Sharing em relação ao modelo de alocação de banda RDM.

Como visto anteriormente, a característica oportunista do modelo de alocação de banda AllocCT-Sharing está em considerar a banda disponível em todas as CTs através do uso integrado de estratégias de compartilhamento "baixa para alta" e "alta para baixa" em um único modelo de alocação de banda. Assim sendo o AllocCT-Sharing compartilha a banda disponível em todas as CTs.

Na próxima subseção, são apresentados os cenários e as campanhas de simulação do algoritmo AllocCT-Sharing, juntamente com as considerações sobre os resultados obtidos, a fim de validar e avaliar as características do AllocCT-Sharing através de uma prova de conceito.

3.4 ALLOCCT-SHARING COM *LOAN CSPF* VERSUS RDM COM CSPF

De maneira geral, a simulação descrita focou na avaliação comparativa do caráter oportunista do modelo de alocação de banda AllocCT-Sharing com o algoritmo de seleção de caminhos baseado em restrições *Loan CSPF* em relação ao modelo de alocação de banda RDM.

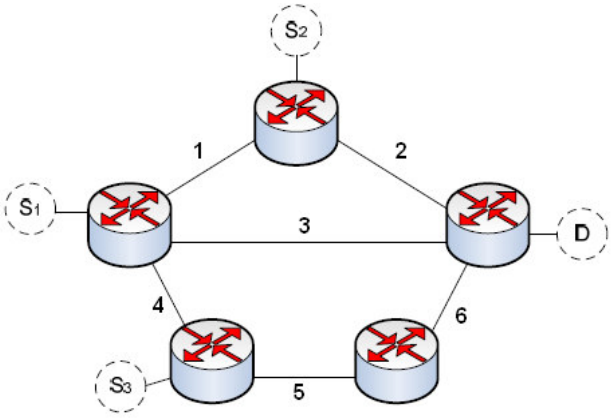
Em particular, optou-se por fazer uma avaliação comparativa em relação ao algoritmo ADAPT-RDM com CSPF, na medida em que este implementa uma versão já avaliada de algoritmo de alocação de banda RDM.

A análise do algoritmo AllocCT-Sharing com CSPF utilizou uma topologia de rede) com três fontes de tráfego (S1, S2 e S3), sendo duas como tráfego de interferência, e um único destino (D), com dois cenários de avaliação.

Cada cenário foi simulado com 05 sementes diferentes e os resultados apresentados são com base na média dos valores obtidos. Para validação, geramos os intervalos de confiança (95%) e desvios padrões[7]. Tivemos o cuidado para não utilizar a mesma semente e o uso do "zero" nas rodadas de simulação.

[7] Os dados resultantes das simulações estão disponíveis em: http://www.rafaelreale.net/alloct_sharing.

Figura 32 – Topologia de Rede – Simulação



Os parâmetros de configuração dos cenários de simulação são os seguintes:

- Enlaces: 622Mbps (STM-4 – SDH);
- Classes de tráfego (CTs) existentes: CT0, CT1 e CT2;
- Restrições de banda (BCs): conforme a tabela 12.

Tabela 12 - Alocação de Banda por Classe de Tráfego (CT)

BC	Max BC (%)	MAX BC (Mbps)	CT por BC
BC0	100	622	CT0+CT1+CT2
BC1	70	435,4	CT1+CT2
BC2	40	248,8	CT2

No cenário de avaliação 01 os modelos foram comparados em uma situação onde o tráfego gerado inicialmente era maior para as classes de tráfego (CTs) de maior prioridade, tentando mostrar assim o caráter oportunista do modelo AllocCT-Sharing em implementar o compartilhamento “baixa para alta”.

No cenário de avaliação 02, os modelos foram comparados com um tráfego maior, alocado para as classes de tráfego (CTs) de menor prioridade em relação às classes de tráfego de maior prioridade. Este é um cenário típico previsto no modelo RDM e, neste sentido, o intuito é de demonstrar que o modelo AllocCT-Sharing possui desempenho equivalente ao modelo RDM nesta situação por também implementar o compartilhamento “alta para baixa”.

3.5 CENÁRIO 1 – ALTO TRÁFEGO PRIORITÁRIO

Neste cenário de simulação as seguintes métricas de desempenho foram comparadas:

- Média da carga dos enlaces;
- Quantidade de preempções;
- Quantidade de bloqueios e
- LSPs atendidos (Mbps).

No caso, compara-se o modelo RDM com CSPF e o modelo AllocCT-Sharing com *Loan CSPF* quando o tráfego de maior prioridade (CT2) possui banda disponível acima da sua restrição de banda e, após um determinado período, gera-se tráfego de competição para as classes CT1 e CT0. Assim sendo, foram definidos os parâmetros de execução da simulação como seguem:

- Intervalo entre chegadas de solicitações de LSPs modelado exponencialmente, tendo como média os seguintes valores para origem do tráfego S1:
  - LSPs – CT0
    - Primeiro LSP gerado com retardo de 500s
    - Gerados a cada 4 s
  - LSPs – CT1
    - Primeiro LSP gerado com retardo de 300 s
    - Gerados a cada 2s
  - LSPs – CT2
    - Gerados a cada 1s
- Para os tráfegos de interferência S2 e S3, o intervalo entre chegadas de solicitações de LSPs é modelado exponencialmente tendo como média os seguintes valores:
  - LSPs – CT0
    - Gerados a cada 20s
    - Primeiro LSP gerado com retardo de 500s
  - LSPs – CT1
    - Gerados a cada 22s
    - Primeiro LSP gerado com retardo de 300s
  - LSPs – CT2
    - Gerados a cada 24s

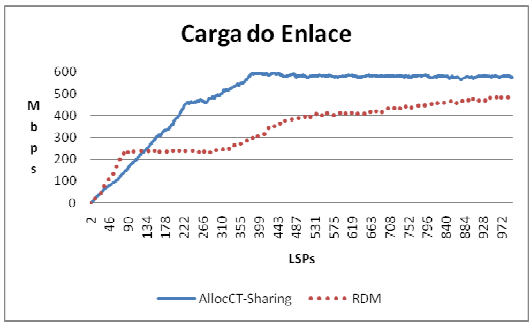
- Demais parâmetros de simulação:
    - LSP - duração do estabelecimento modelada exponencialmente - média de 150 segundos, provocando a saturação do enlace;
    - LSP – banda - distribuída uniformemente entre 05Mbps e 20Mbps e
    - Número de LSPs – 1.000

O critério de parada de simulação utilizado foi o número de LSPs gerados, pois o cenário se repetia para valores superiores devido a todas as CTs utilizarem o máximo permitido para suas restrições.

Seguem uma análise e avaliação dos resultados obtidos:

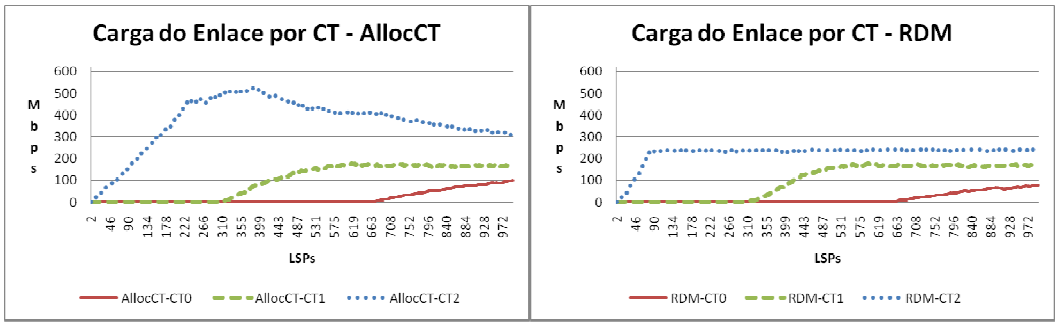
Na simulação executada, apenas existem LSPs da CT2 sendo geradas até o instante 300 segundos. A figura 33 mostra que o modelo RDM limita a carga média dos enlaces ao valor de 248,8Mbps, configuração do seu BC2. No modelo RDM, somente após a chegada de LSPs de CT1 e CT0 tem-se que a carga média dos enlaces assume valores maiores que 248,8Mbps. Já o modelo AllocCT-Sharing permite que seja utilizada a banda livre das demais classes de tráfego, de forma oportunista, como empréstimo para a CT2 e, depois, passa a devolvê-la com a chegada de LSPs das CT0 e CT1.

Figura 33 – Média da Carga dos Enlaces



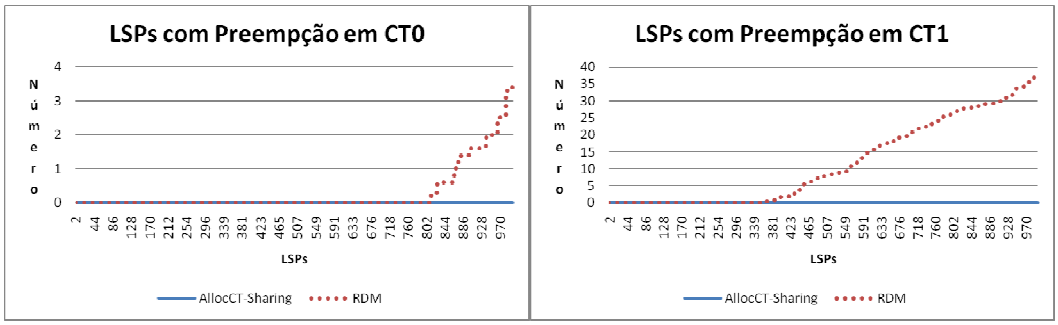
A carga do enlace por CT (figura 34) mostra que o uso oportunista da banda livre no modelo AllocCT-Sharing e depois a devolução no instante que os LSP ligados ao CT0 e CT1, respectivamente, necessitam de banda. É importante observar que, no modelo AllocCT-Sharing, após CT2 devolver a banda emprestada, as cargas do enlace por CT ficam bastante semelhantes ao modelo RDM, demonstrando que o mesmo converge para o modelo RDM quando as CTs passam a solicitar a totalidade de suas bandas configuradas.

Figura 34 – Média da Carga do Enlace por CT



Um aspecto importante a ser analisado para os modelos de alocação de banda é o volume de preempções decorrente. Conforme verificado na figura 35, as preempções nos modelos só começaram instantes após a chegada dos tráfegos competitivos das CT0 e CT1 devido à concorrência entre as CTs. Mesmo com banda disponível no enlace, verificou-se que o modelo RDM gerou preempções desnecessárias por prioridade em CT1, ao chegar LSPs de CT2, para atender o limite de BC1. As preempções em CT0 foram por prioridade. Já no modelo AllocCT-Sharing, sua característica oportunista reduziu o número de preempções ocorridas em CT1, já que o enlace que não é saturado não é motivo para preempções, permanecendo apenas as preempções ocorridas após a saturação do enlace. Obviamente, é importante ressaltar que a simulação foi parametrizada de forma a demonstrar esta característica do novo modelo proposto.

Figura 35 – Preempção por Classe de Tráfego (CTs)

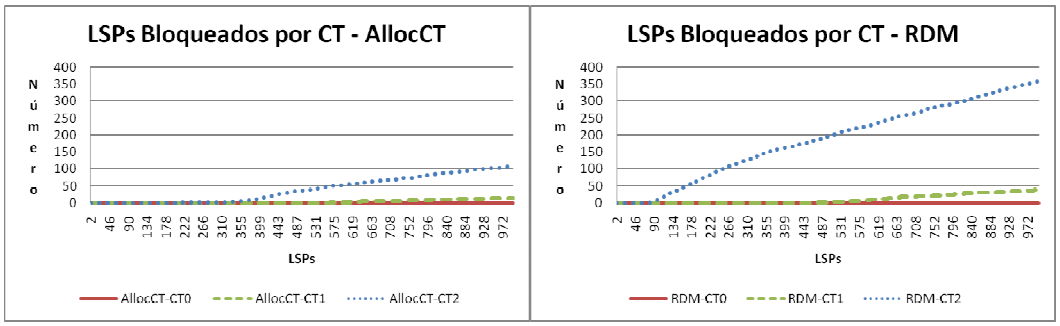


Numa situação de tráfego real, a vantagem inerente ao modelo AllocCT-Sharing surge apenas quando a solicitação dinâmica para LSPs do tráfego prioritário fica acima do valor especificado para sua CT. Em resumo, o algoritmo AllocCT-Sharing pode prover benefícios (mais banda) para as aplicações prioritárias quando, no dinamismo do tráfego real, esta

aplicação solicitar banda e as aplicações de baixa prioridade não estejam fazendo uso do recurso (banda).

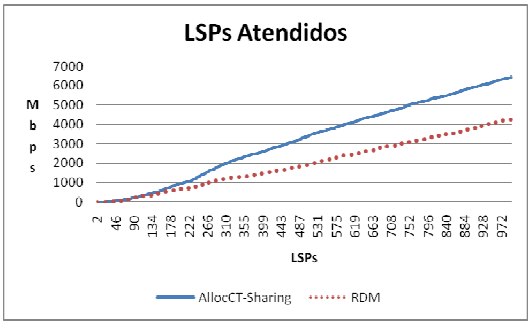
A figura 36 ilustra o comportamento dos modelos com relação aos bloqueios por classe de tráfego. Devido à banda disponível para a CT2 ter sido limitada a 248,8Mbps, e devido à configuração da restrição de banda para o BC2 no modelo RDM para este cenário, todos os LSPs que chegaram após esse limite ser atingido foram bloqueados, mesmo havendo banda disponível no enlace. No modelo AllocCT-Sharing os bloqueios somente começaram a ocorrer quando o enlace saturou, ou seja, não havia banda disponível.

Figura 36 – Bloqueios por Classe de Tráfego (CTs)



Devido ao menor bloqueio e menor número de preempções de LSPs no AllocCT-Sharing para o cenário de simulação considerado, houve um valor consideravelmente maior de LSP atendidos. A figura 37 demonstra que, neste cenário, o AllocCT-Sharing permite uma maior utilização dos enlaces de rede, aproximadamente 35% (2169Mbps). Consideraram-se como LSPs atendidos os que foram estabelecidos e concluídos sem preempção.

Figura 37 – LSPs Atendidos (Mbps)



Um aspecto não considerado neste estudo inicial do algoritmo AllocCT-Sharing foi o impacto para as aplicações prioritárias considerando que alguns LSPs devem ser desalocados para a efetiva liberação de banda. Em efeito, considera-se inicialmente que as aplicações com LSPs desalocáveis possam estar, atreladas às aplicações prioritárias e, ao mesmo tempo, flexíveis de

forma que a aplicação tanto possa se beneficiar de banda extra disponível, como mitigar os efeitos de sua eventual redução no cenário indicado de conflito.

3.6 CENÁRIO 2 - BAIXO TRÁFEGO PRIORITÁRIO

Neste cenário de simulação as seguintes métricas de desempenho foram comparadas:

- Média da carga dos enlaces;
- Quantidade de preempções;
- Quantidade de bloqueios e
- LSPs atendidos (Mbps).

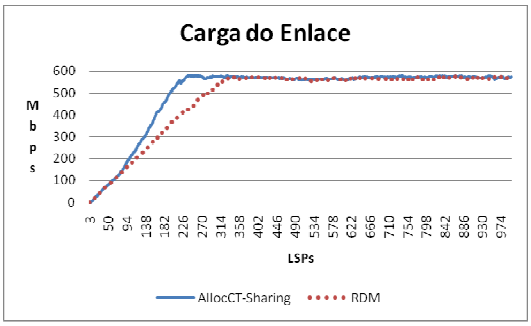
No caso, compara-se o modelo RDM com CSPF e o modelo AllocCT-Sharing com *Loan CSPF* quando é gerado um tráfego maior alocado para as classes de tráfego (CTs) de menor prioridade em relação às classes de tráfego de maior prioridade. Este é um cenário típico previsto no modelo RDM. Assim sendo, foram definidos os parâmetros de execução da simulação como segue:

- Intervalo entre chegadas de solicitações de LSPs, modelado exponencialmente tendo como média os seguintes valores para origem do tráfego S1:
  - LSPs – CT0 – 1 s
  - LSPs – CT1 - 2 s
  - LSPs – CT2 - 4 s
- Para os tráfegos de interferência S2 e S3 o intervalo entre chegadas de solicitações de LSPs modelado exponencialmente tendo como média os seguintes valores:
  - LSPs – CT0 – 20 s
  - LSPs – CT1 - 22 s
  - LSPs – CT2 - 24 s
- Demais parâmetros de simulação:
  - LSP - duração do estabelecimento modelada exponencialmente - média de 170 segundos, provocando a saturação do enlace;
  - LSP – banda - distribuída uniformemente entre 05Mbps e 20Mbps e
  - Número de LSPs – 1.000.

O critério de parada de simulação utilizado foi o número de LSPs gerados, pois o cenário se repetia para valores superiores devido a todas as CTs utilizarem o máximo permitido para suas restrições. Seguem os resultados obtidos.

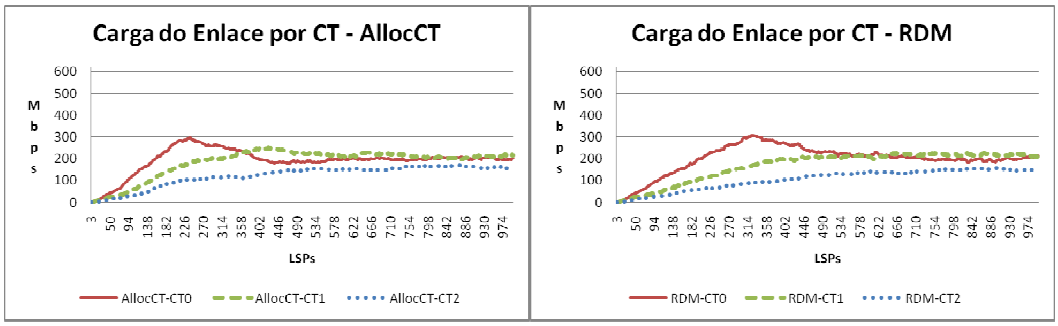
Na simulação executada (Figura 38) identifica-se que o modelo RDM e o AllocCT-Sharing possuem utilização do enlace semelhantes. O AllocCT-Sharing obteve vantagem nos primeiros instantes devido à sua premissa de permitir o livre uso do enlace até que o mesmo sature.

Figura 38 – Carga Média do Enlace



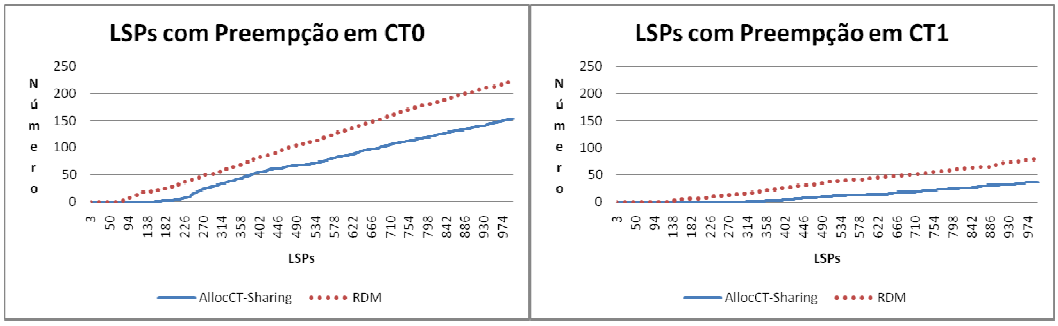
A carga do enlace por CT (figura 39) possui comportamento semelhante apenas diferenciando por alguns picos permitidos no modelo AllocCT-Sharing implicando em uma maior flexibilidade do modelo.

Figura 39 – Carga Média do Enlace por CT



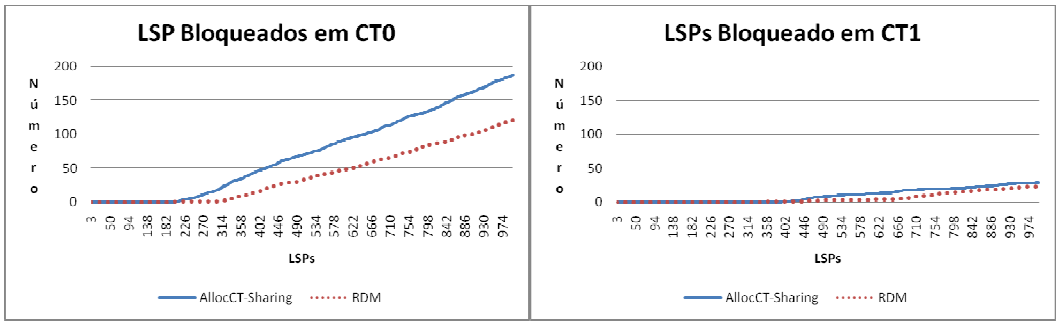
Conforme observado na figura 40, o AllocCT-Sharing obteve, neste cenário, um desempenho levemente melhor com 56 preempções a menos que o modelo RDM, otimizando principalmente as preempções em CT1. Registramos que o modelo RDM causou a preempção desnecessária de LSPs, para atender a restrição do modelo, mesmo havendo banda livre. O principal motivo das preempções em ambos os modelos foi a saturação do enlace que levou à preempções por prioridade.

Figura 40 – Preempção por Classe de Tráfego (CTs)



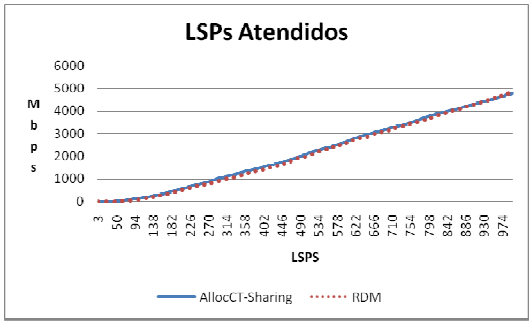
A figura 41 ilustra o comportamento dos modelos comparados com relação aos bloqueios por classes de tráfego. Em ambos os modelos, o principal motivo de bloqueio foi a saturação do enlace. Devido ao maior número de LSPs atendidos e o menor número de LSPs com preempção, o modelo AllocCT-Sharing trabalhou sempre no máximo permitido pelo enlace. Esse comportamento levou a 69 LSPs bloqueados a mais em relação ao modelo RDM.

Figura 41 – Bloqueios por Classe de Tráfego (CTs)



Em relação aos LSPs atendidos, os modelos obtiveram desempenho semelhante como pode ser observado na figura 42.

Figura 42 – LSPs Atendidos



Nos dois cenários apresentados, as SLAs são garantidas, pois em situação de efetiva concorrência por banda na rede, o modelo de alocação de banda AllocCT-Sharing converge os usos de banda pelas CTs para o valor configurado de restrição de banda (BC) pelo gerente de rede como pode ser observado na figura 34 e na figura 39.

O AllocCT-Sharing e o RDM têm comportamento similar apenas no cenário onde são gerados mais LSPs de CTs menos prioritárias do que de CTs mais prioritárias. Ou seja, o AllocCT-Sharing "mantém" o comportamento já demonstrado pelo RDM neste segundo cenário (uso de banda de CTs de prioridades superiores por CTs de prioridades inferiores) (ADAMI *et al*, 2007; PINTO NETO; MARTINS, 2008a; PINTO NETO; MARTINS, 2008b; PINTO NETO 2008).

No cenário oposto, o modelo AllocCT-Sharing apresenta melhor desempenho, pois permite que LSPs de alta usem banda livre de LSPs de baixa prioridade. O RDM não permite esta alternativa.

Assim sendo, AllocCT-Sharing mantém o desempenho do RDM para um perfil de tráfego e melhora para o perfil de tráfego oposto. Entende-se então que o resultado global é melhor, e a utilização global dos enlaces fica bem melhorada conforme cenários e simulações apresentadas.

Como forma de implementar a prova de conceito, optamos em utilizar o simulador BAMSim (*Bandwidth Allocation Model Simulator*) proposto em Pinto Neto (2008), pois o mesmo já possui implementadas as funcionalidades de uma rede DS-TE habilitada com o modelo de restrição de banda RDM e todo arcabouço necessário para criação de novos modelos de alocação de banda e algoritmos de seleção de caminhos para estas redes.

Assim sendo, a estratégia de adoção do simulador utilizado na avaliação por prova de conceito consistiu em adotar o BAMSim e fazer todos os ajustes e complementos de codificação necessária para o novo modelo de alocação de banda proposto, o AllocCT-Sharing.

3.7 O SIMULADOR BAMSIM

O BAMSim é um simulador de propósito específico (MACDOUGALL, 1987) desenvolvido em C e baseado na execução de um conjunto de eventos discretos no tempo. O BAMSim é mantido pelo grupo de IP&QoS da Universidade Salvador.

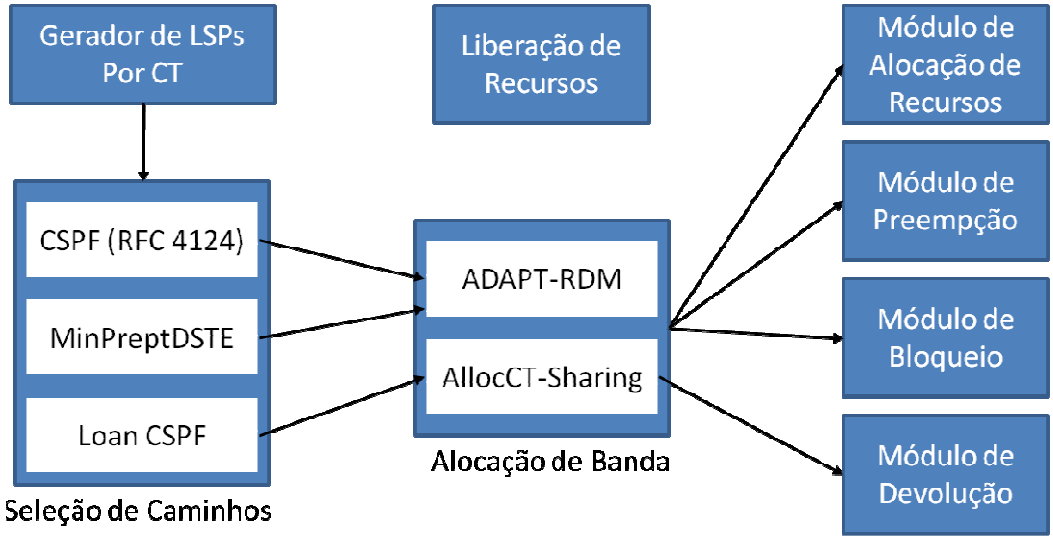
Este trabalho de dissertação contribui com três novos módulos ao BAMSim:

- O módulo *Loan CSPF* - Implementa o algoritmo de seleção de caminhos CSPF com o conhecimento de empréstimos como descrito no capítulo anterior.

- O módulo AllocCT-Sharing - Implementa o modelo oportunista de alocação de banda AllocCT-Sharing.
- O módulo de "devolução" - Implementa a devolução de empréstimos na medida em que os empréstimos possuem características particulares que o conceito de preempção não atende por si só.

Os principais módulos do BAMSim já com a adição dos três novos módulos propostos neste trabalho (*Loan CSPF*, AllocCT-Sharing e devolução de "empréstimos") podem ser observados na figura 43.

Figura 43 – Módulos do BAMSim



Informações adicionais referentes ao simulador são apresentadas no Anexo A.

4 CONSIDERAÇÕES FINAIS

A demanda por recursos de rede vem crescendo vertiginosamente, motivada pelo crescimento do número de usuários e computadores conectados às redes. Neste cenário, as redes multisserviço destacam-se como alternativa para o atendimento de serviços distintos e diferenciados para usuários de maneira geral. As redes multiserviços oferecem uma única infraestrutura de rede capaz de atender diversos serviços com necessidades distintas de Qualidade de Serviço (QoS). Uma alternativa para implementação de redes multiserviços é a utilização da arquitetura DS-TE (*DiffServ Aware MPLS Traffic Engineering*), por facilitar e estruturar a implantação de mecanismos de alocação de banda; desta forma, facilita a utilização de estratégias de QoS e engenharia de tráfego para classes de tráfego (CT).

Para a melhor eficiência na utilização dos recursos em uma rede DS-TE, é necessária a implementação de modelos de alocação de banda e algoritmos de seleção de banda baseados em restrições. A implementação de um modelo de alocação de banda permite regular a relação de isolamento e/ou compartilhamento entre as classes de tráfego (CT) configuradas na rede, diretamente refletindo em seu desempenho como um todo.

Esta dissertação propôs, desenvolveu e validou um novo algoritmo de alocação de banda e a sua integração ao algoritmo de seleção de caminhos CSPF, possibilitando assim, uma utilização mais eficiente dos enlaces numa rede DS-TE para determinados cenários de distribuição de tráfego.

O algoritmo AllocCT-Sharing é um modelo oportunista de alocação de banda, na qual os LSPs associados às classes de tráfego de maior prioridade podem utilizar de forma circunstancial a banda disponível e não utilizada por classes de tráfego menos prioritárias e vice-versa. Desta forma o AllocCT-Sharing criou efetivamente o conceito de "empréstimo" de banda com um mecanismo de "devolução" da mesma quando solicitado ou requerido pela classe de tráfego proprietária.

Em efetivo, foi implementado, com base na RFC4124, uma extensão do algoritmo CSPF que considera o conceito de "empréstimo" (AllocCT-Sharing), através do qual foi denominado "*Loan CSPF*". A proposta como um todo foi validada (prova de conceitos).

Com relação ao novo modelo de alocação proposto (AllocCT-Sharing), observou-se na validação do mesmo as seguintes características e comportamentos:

1. Em qualquer cenário em que não haja saturação do enlace, o AllocCT-Sharing com "*Loan CSPF*" permite um maior compartilhamento entre as CTs e uma melhor utilização da banda disponível no enlace.
2. Nos cenários onde existe a saturação do enlace, o AllocCT-Sharing com CSPF comporta-se de forma semelhante aos demais modelos de alocação de banda "alta para baixa" como, por exemplo, o RDM com CSPF.
3. As preempções e bloqueios só ocorrem quando da efetiva disputa por recursos (banda alocada).
4. As simulações indicam também que, estatisticamente, nem todo LSP que solicita banda emprestada precisa devolver (preempção) antes do seu encerramento e, assim sendo, tem-se novamente um sinalizador da melhor e mais flexível utilização da banda de enlace disponível.

Concluímos que o AllocCT-Sharing amplia o compartilhamento entre CTs, implicando desta forma numa melhor utilização dos enlaces e na redução de preempções e bloqueios, quando comparado ao modelo RDM. De forma mais genérica, as características do modelo AllocCT-Sharing proporcionam às aplicações de uma rede multisserviços a possibilidade de melhorar seu serviço, de forma oportunista, sempre que haja banda disponível no enlace.

4.1 TRABALHOS FUTUROS

Em termos dos trabalhos futuros, pretende-se simular o comportamento dos algoritmos propostos em uma matriz de tráfego real; desenvolver um algoritmo de seleção de caminhos que otimizem as características oportunistas do AllocCT-Sharing, implicando em uma utilização ainda mais eficiente dos enlaces da rede; aplicar políticas diretamente nos LSPs, como limiar aceitável de redução da banda inicialmente solicitada, com o objetivo de permitir uma maior elasticidade da rede a partir do seu ponto de menor granularidade que são os LSPs; implementar uma solução descentralizada dos algoritmos propostos, e;. desenvolver fatores probabilísticos para redução de preempções e devolução dos empréstimos de acordo com o nível de saturação do enlace

REFERÊNCIAS

ADAMI, D. et al. G-RDM: a new bandwidth constraints model for DS-TE networks. In: IEEE GLOBECOM, 2007. **Proceedings**… 2007.

ADAMI, D. et al. A New NS2 simulation module for bandwidth, 2008. In: IEEE ICC, 2008. **Proceedings…** 2008.

ADAMI, D. et al. NS2 Extensions for the Simulation of RDM and G-RDM in DS-TE networks. In: IEEE ICC, 2010. **Proceedings**…2010.

ANDERSSON, L.; SWALLOW, G. The Multiprotocol Label Switching (MPLS) Working Group decision on MPLS signaling protocols. **IETF, RFC 3468**, 2003.

ARMITAGE, G. **Quality of Service in IP Networks**. 1. ed. [S.l.]: New Riders Publishing, 2000.

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

AWDUCHE, D. MPLS and Traffic Engineering in IP Networks. **IEEE Communications Magazine,** v. 37, p. 42-47, ISSN: 0163-6804, 1999b.

AWDUCHE, D; et al. Overview and Principles of Internet Traffic Engineering. **IETF, RFC 3272**, 2002.

AWDUCHE, D et al. RSVP-TE: Extensions to RSVP for LSP Tunnels. **IETF, RFC3209,** 2001.

BLAKE, S. et al. An Architecture for Differentiated Services. **RFC 2475,** 1998. ENNE, A. TCP/IP sobre MPLS. 1. ed. Rio de Janeiro: Ciência Moderna Ltda., 2009.

HEINANEN, J. et al. Assured Forwarding PHB Group. **IETF, RFC 2597**, 1999.

JACOBSON, V.; NICHOLS, K.; PODURI, K. An Expedited Forwarding PHB. **IETF, RFC 2598**, 1999.

JAMOUSSI, B et al. Constraint-Based LSP Setup using LDP. **IETF, RFC 3212,** 2002.

KAMIENSKI, C. A.; SADOK, D. Qualidade de Serviço na Internet. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES (SBRC'2000), 18., 2000. Belo Horizonte. **Anais...** 2000.

KUROSE, J.; ROSS, K. **Redes de computadores e a Internet**: uma abordagem top-down. São Paulo: Pearson Education do Brasil, 2010.

LAI, W.; Bandwidth Constraints Models for DiffServ-aware MPLS Traffic Engineering: Performance Evaluation. **IETF, RFC 4128**, 2005.

LE FAUCHEUR, F. Russian Dolls Bandwidth Constraints Model for DiffServ-Aware MPLS Traffic Engineering. **IETF, RFC 4127,** 2005a.

LE FAUCHEUR, F. Protocol Extensions for Support of DiffServ-aware MPLS Traffic Engineering. **IETF, RFC 4124**, 2005b.

LE FAUCHEUR, F.; LAI, W. Maximum Allocation Bandwidth Constraints Model for DiffServ-Aware MPLS Traffic Engineering. **IETF, RFC 4125**, 2005.

LE FAUCHEUR, F.; LAI, W. Requirements for Support of Differentiated Services-Aware MPLS Traffic Engineering. **IETF, RFC 3564**, 2003.

LE FAUCHEUR, F. et al. Multi-Protocol Label Switching (MPLS) Support of Differentiated Services. **IETF, RFC 3270**, 2002.

MACDOUGALL, M. H. **Simulating Computer Systems Techniques and Tools**. Massachusetts: The Massachusetts Institute of Technology Press, 1987.

MARTINS, Joberto. **Qualidade de Serviço em redes IP**: princípios básicos, parâmetros e mecanismos. Disponível em: <http://www.jsmnet.com>. Acesso em: 22 ago. 2011.

MARTINS, J. et al. **Managing IP Networks:** Challenges and Opportunities 1. ed. New Jersey - USA: John Wiley & Sons/ IEEE Press, 2003. v. 1.

MINEI, I.; LUCEK, J. **MPLS-Enabled Applications:** emerging developments and new technologies. [S.l.]: John Wiley & Sons Ltd, 2005.

MIRAS, D. A **Survey on Network QoS Needs of Advanced Internet Applications, Internet2 QoS Working Group.** [S.l.]: [s.n.], 2002.

MOREIRA, M. D. D. et al. Internet do Futuro: um novo horizonte. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES - SBRC'2009. 2009. Rio de Janeiro. **Anais...** 2009. (Minicursos).

MONTEIRO, A. S. M.; SAMPAIO, L.; FIGUEREDO, M. **Qualidade de Serviço:** Diagnóstico e Alternativas, Relatório Parcial do Grupo de Trabalho de Qualidade de Serviço (GT-QoS) da RNP2. [S.l.]: [s.n.], 2002.

OSBORNE, E.; SIMBA, A. **Engenharia de Tráfego com MPLS – Projeto, Configuração e Gerenciamento do MPLS-TE para Otimização de Desempenho de Rede, Cisco System.** São Paulo:Campus Ltda, 2003.

PINTO NETO, W.; MARTINS, J.; BRITO, S. Algoritmos de Seleção de Caminho e Gerenciamento de Banda Compartilhada conforme ao Modelo RDM para Classes de Tráfego em Rede DS-TE. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS - SBRC, 26., 2008. Rio de Janeiro. **Anais…** 2008.

PINTO NETO, W.; MARTINS, J. A RDM-like Bandwidth Management Algorithm for Traffic Engineering with DiffServ and MPLS Support. In: INTERNATIONAL CONFERENCE ON TELECOMMUNICATIONS - ICT, 15., 2008. St. Peterburg, Rússia. **Proceedings…** 2008a.

PINTO NETO, W.; MARTINS, J. Adapt-RDM - A Bandwidth Management Algorithm suitable for DiffServ Services Aware Traffic Engineering. In: IEEE/IFIP NETWORK OPERATIONS & MANAGEMENT SYMPOSIUM, NOMS, 2008. Salvador. **Anais...** 2008b.

PINTO NETO, W. **Proposta e Avaliação**: algoritmos para a Gerência de Banda e Seleção de Caminhos em Redes MPLS (DS-TE). 2008. Dissertação (Mestrado Acadêmico em Sistemas e Computação) - Universidade Salvador UNIFACS, Salvador, 2008.

ROSEN, E.; VISWANATHAN, A; CALLON, R.; Multiprotocol Label Switching Architecture. **IETF, RFC 3031**, 2001.

REALE, R; PINTO NETO, W.; MARTINS, J. Modelo de Alocação de Banda com Compartilhamento Oportunista entre Classes de Tráfego e Aplicações em Redes Multiserviço, In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS - SBRC, 29., 2011. Campo Grande-MS. **Anais...** 2011.

SANTANA, S. **Proposta de Referência para Projetos de Qualidade de Serviço (QoS) em Redes Corporativas.** 2006. Dissertação (Mestrado em Redes de Computadores) - Universidade Salvador UNIFACS, Salvador, 2006.

SCOGLIO, C. et al. TEAM: A Traffic Engineering Automated Manager for DiffServ-Based MPLS Networks. **IEEE Communications Magazine**, p. 134-145, 2004.

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

TANENBAUM, A. S. **Redes de Computadores**. 4. ed. São Paulo: Campus, 2003.

TONG, S.; YANG, O. Bandwidth Management for Supporting Differentiated Service Aware **Traffic Engineering, IEEE Trans**, v.18, n.9, p.1320-1331, 2007

TRIMINTZIOS, P. et al. A Management and Control Architecture for Providing IP Differenciated Services in MPLS-Based Networks. **Ieee Communications Magazine**, 2001. P. 80-88.

ANEXO A – BAMSIM (Bandwidth Allocation Model Simulator)

O simulador de propósito específico BAMSim (*Bandwidth Allocation Model Simulator*) foi desenvolvido no trabalho de dissertação intitulado "Proposta e Avaliação: Algoritmos para a Gerencia de Banda e Seleção de Caminhos em Redes MPLS (DS-TE)" por Walter da Costa Pinto Neto orientado por Prof. Joberto Sérgio Barbosa Martins e hoje é mantido pelo grupo de pesquisa IP&QoS da Universidade Salvador.

Este trabalho de dissertação contribuiu com dois novos algoritmos ao BAMSim: o algoritmo de seleção de caminhos baseado em restrições "*Loan CSPF*" e o modelo de alocação de banda AllocCT-Sharing. Associados a estes foram realizadas alterações no BAMSim no intuito de incluir os conceitos de "empréstimos" e "devoluções" necessários para a implementação dos mesmos.

Abaixo apresentamos o atual funcionamento do BAMSim.

BAMSIM VERSÃO 2.0

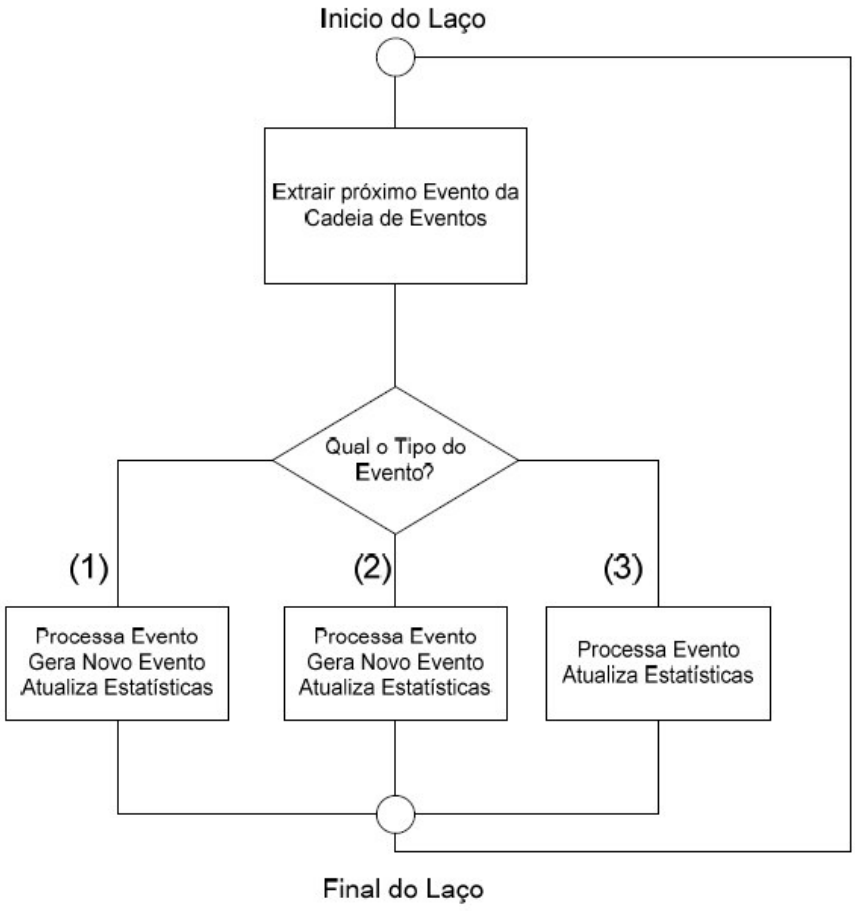
O simulador de propósito específico BAMSim está baseado na execução de um conjunto de eventos discretos no tempo (MACDOUGALL, 1987). A ideia é que a partir da "execução cíclica" de um conjunto de eventos ordenados em uma lista (denominada cadeia de eventos) em ordem cronológica, seja possível representar computacionalmente e realizar a modelagem desejada de uma rede DS-TE utilizando o modelo RDM ou AllocCT-Sharing.

A inserção e retirada de eventos da cadeia de execução de eventos caracteriza o processo de funcionamento da máquina de eventos do simulador. A implementação do simulador de propósito específico desenvolvido é baseada na filosofia de funcionamento de execução da cadeia de eventos da linguagem de simulação SMPL, especificada em Macdougall (1987). O SMPL é uma extensão funcional (biblioteca de funções) da linguagem C de propósito geral, adequado para modelos de simulação pequenos ou médios. Uma ilustração do funcionamento da execução da máquina de eventos é mostrada na figura 44 a seguir.

Durante a simulação um LSP é tratado como um cliente. A partir do processamento cíclico de um conjunto de eventos discretos associados ao LSP, é possível simular o sistema desejado. Um evento a ser processado é retirado do topo da cadeia de eventos, e dependendo do tipo ao qual pertença, passará por determinado processamento. Após o processamento de um evento, um outro tipo de evento é gerado se existir o encadeamento de eventos. No caso do exemplo da figura 44, após o processamento dos eventos tipo 1 e 2, é gerado um outro tipo de evento.

Após o processamento do evento tipo 3 não há encadeamento de eventos. Após o processamento de cada evento as estatísticas de simulação são atualizadas.

Figura 44 - Máquina de Eventos do Simulador



O BAMSim utiliza-se do gerador de números aleatórios da "srand()" implementado como biblioteca da linguagem C. A geração de variáveis aleatórias segundo as distribuições Uniforme e Exponencial são implementadas utilizando-se dos fragmentos de códigos da linguagem de simulação SMPL definida em Macdougall (1987), cujo trechos dos códigos são mostrados no ANEXO B desse trabalho.

CONFIGURAÇÃO DA SIMULAÇÃO NO BAMSIM

Esta seção descreve as etapas para preparar uma simulação utilizando o BAMSim para simular o comportamento de uma rede DS-TE utilizando o modelo RDM ou AllocCT-Sharing.

O preparo da simulação passa pela configuração da topologia da rede e das particularidades de uma rede DS-TE utilizando o modelo RDM ou AllocCT-Sharing. Ele está associado a um conjunto de etapas, descritas a seguir:

Primeira Etapa: Definição da Topologia e das Configurações DS-TE (RDM ou AllocCT-Sharing)

Construção da topologia de rede que se deseja simular criando os nós (roteadores) e os enlaces que conectam os nós do modelo;

- Associação dos geradores de LSPs aos respectivos nós aos quais eles estão associados no modelo;
- Definição do número de CTs configuradas em rede;
- Definição das quantidades máximas dos BCs configurados em rede.

Segunda Etapa: Inicialização de LSPs por CT

Para a utilização da biblioteca de funções implementada, é necessário a inicialização dos LSPs por CT para cada nó origem de tráfego na rede. A inicialização feita no programa principal consiste da geração manual de LSPs no tempo de simulação 0 (zero) antes do início da execução da simulação (pré-requisito para o início do funcionamento da máquina de eventos). Em cada nó origem de tráfego na rede é inicializada um LSP para cada CT configurada.

Terceira Etapa: Execução do Laço de Simulação

Retirada do evento atual a ser processado do final da cadeia de eventos. Três eventos são possíveis: (evento 1) Geração de um LSP; (evento 2) Tentativa de Estabelecimento de LSP; (evento 3) Desativação de LSP. Esses eventos são detalhados na seção seguinte.

- Processamento do evento retirado do topo da cadeia de eventos;
- Geração de um novo evento para a cadeia de eventos:
    - Após a execução do evento 1 (Geração de LSP) dois novos eventos são gerados. É gerado um evento 2 (Tentativa de Estabelecimento de LSP) para a tentativa de estabelecimento do LSP gerado, e é gerado um novo evento 1 que define quando um próximo LSP nas mesmas condições do LSP ao qual foi posta em tentativa de estabelecimento deverá ser gerado.

- Após a execução do evento 2 (Tentativa de Estabelecimento de LSP) em caso de sucesso no estabelecimento do LSP, é gerado um evento 3 (Desestabelecimento do LSP) com o tempo no qual a LSP deverá permanecer estabelecida (i.e., tempo de vida da LSP). Em caso de insucesso no estabelecimento do LSP nenhum outro evento é gerado, sendo contabilizado o bloqueio do LSP por falta de recursos. As estatísticas dos enlaces, das CTs e das BCs são atualizadas a cada estabelecimento de LSP.
- Após a execução do evento 3 (Desestabelecimento de LSP) o LSP é desestabelecido e as estatísticas dos enlaces, das CTs e das BCs são atualizadas. Nenhum outro evento é gerado, após a execução de um evento 3.

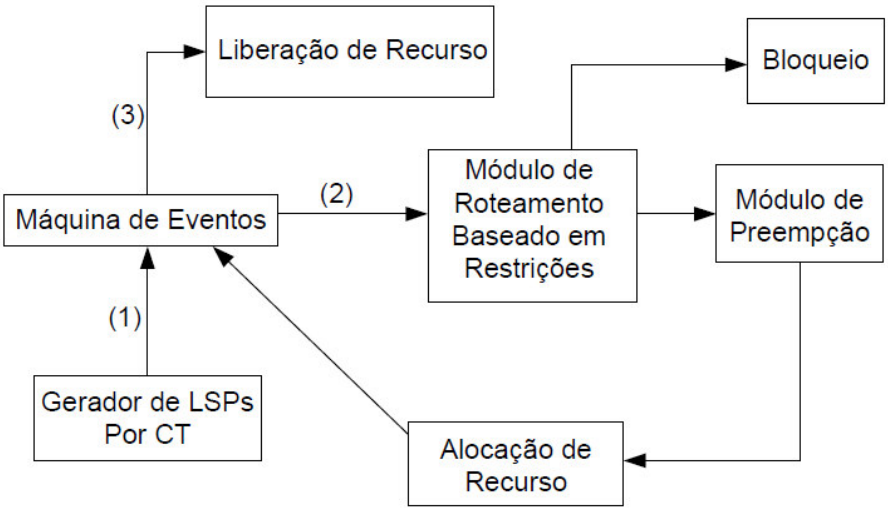
Quarta Etapa: Coleta das Estatísticas de Simulação

Ao final da execução do laço (final da simulação), o programa coleta os dados de desempenho de interesse obtidos para análise pós-simulação.

EVENTOS E MÓDULOS DA SIMULAÇÃO

Os eventos que são passíveis de ocorrência e os módulos do simulador juntamente com a filosofia de funcionamento implementada são detalhados nesta seção. Um fluxograma para facilitar o entendimento é mostrado na figura 45:

Figura 45 - Eventos, Módulos e Principais Etapas da Simulação



EVENTO 1: GERAÇÃO DE UM NOVO LSP PARA A REDE

Neste evento é realizada a geração de LSPs associado a um determinado nó (roteador) da rede. Os LSPs são gerados por CT com diferente tempos de interchegada baseados na distribuição exponencial (Macdougall, 1987). O tempo médio de interchegada na geração de LSPs pode variar a depender da configuração de cada uma das CTs configuradas em rede.

As seguintes definições são necessárias para a execução do Evento 1 (geração de LSPs), considerando a forma como a implementação foi realizada e as particularidades de interesse de uma rede DS-TE utilizando o modelo RDM ou AllocCT-Sharing:

- Definição do nó de origem do LSP na rede: Os LSPs terão como possíveis origens os nós geradores de LSPs que forem definidos na simulação;
- Definição do nó de destino do LSP na rede: Os LSPs terão como possíveis destinos os nós sorvedouros de LSPs que forem definidos na simulação;
- Associação à Matriz de Caminhos: O LSP é associado a uma tabela que contém os caminhos disponíveis entre o nó de origem e o nó de destino através de um indexador (A Matriz de Caminhos está armazenada na entidade gerenciadora);
- Banda do LSP: Definida com base na distribuição uniforme, onde existe um valor de banda mínimo e um valor de banda máximo especificado para cada CT configurado em rede;
- Prioridade do LSP: Define a prioridade do LSP para o caso da necessidade futura de preempção;

EVENTO 2: TENTATIVA DE ESTABELECIMENTO DE LSP

Nesse evento é tentado o estabelecimento de uma nova requisição de LSP que foi gerada. A partir desse evento são executados três módulos principais: O módulo de seleção de caminhos, o módulo de verificação de necessidade de preempção de banda e o módulo de alocação de recursos.

O **módulo de seleção de caminhos** é o responsável pelo cálculo do "melhor caminho", a depender das exigências do algoritmo de roteamento baseado em restrições que estiver sendo utilizado (CSPF, "*Loan CSPF*" ou MinPreptDSTE), dentro do conjunto de caminhos possíveis desde o nó de origem até o nó de destino da nova requisição de LSP. Se não existir um caminho que atenda os requisitos na nova requisição de LSP associada a determinado CT, o mesmo será bloqueado. São parâmetros do módulo de seleção de caminhos:

- Nó origem e nó de destino da nova requisição;

- Apontador para a Matriz de Caminhos possíveis entre um nó de origem e um nó de destino.
- Banda exigida e CT correspondente à nova requisição de LSP;
- Banda máxima permitida aos BCs configurado em rede;
- Capacidade não utilizada dos enlaces da rede por BC. Em outras palavras, a banda atual ocupada por cada BC configurado em rede.

O **módulo de verificação da necessidade de preempção de banda** é o responsável por checar se existe a necessidade de adaptação de banda em CTs hierarquicamente inferiores ou superiores à CT da nova requisição de LSP. A verificação é feita em cada enlace do caminho escolhido, visando manter as configurações do modelo RDM ou AllocCT-Sharing. A adaptação é feita através da preempção ou devoluções (caso seja usado o modelo AllocCT-Sharing) de LSPs estabelecidas em rede em conformidade com o modelo de alocação de banda selecionado. São parâmetros do módulo de verificação da necessidade de preempção:

- Caminho escolhido pelo módulo de seleção de caminho;
- Banda exigida e CT correspondente à nova requisição de LSP;
- Banda máxima permitida aos BCs configurado em rede;
- Banda atual ocupada por cada BC configurado em rede;
- Fator w(LSP) da nova requisição de LSPs visando compará-lo com o Fator w(LSP) de LSPs já estabelecidas em caso de necessidade de preempção de LSPs em determinado enlace;

O **módulo de alocação de recurso** é o responsável por simular o estabelecimento de fato da nova requisição de LSP. Atualiza as estatísticas da entidade gerenciadora de banda pós-estabelecimento. A atualização das variáveis de controle da entidade gerenciadora de banda representa no simulador o estabelecimento de fato do LSP, e a posse dos recursos dos enlaces pelo mesmo. Algumas atualizações na entidade gerenciadora são feitas para representar o estabelecimento no simulador de um LSP:

- Armazenamento do novo LSP estabelecido na base de dados de LSPs estabelecidos agrupados por CT, em todos os enlaces associados ao novo caminho;
- Atualização da banda dos enlaces no caminho escolhido para a nova requisição de LSP;
- Atualização da banda atual ocupada por cada BC configurado em rede nos enlaces do caminho da nova requisição de LSP;

- Tempo em que o LSP permanecerá estabelecido (tempo de vida da LSP).

EVENTO 3: DESATIVAÇÃO DE UMA LSP

Nesse evento é realizada a desativação de um LSP. O Evento 3 pode ser executado em três situações distintas na implementação do simulador.

A primeira é a desativação de um LSP devido ao estouro de seu "tempo de vida", ou, em outras palavras, quando o tempo de duração de um LSP se esgota. O tempo de duração do LSP é definido no momento em que ele é estabelecido (Evento 2) na simulação. As seguintes atualizações na entidade gerenciadora após o desestabelecimento de um LSP devido ao estouro de seu "tempo de vida" devem ser realizadas:

- Retirada do LSP da base de dados de LSPs estabelecidos agrupados por CT, em todos os enlaces as quais as mesmas estiverem associadas;
- Atualização da variável A_BC[n] (banda atual ocupada pelo determinada restrição de banda) nos enlaces do caminho do LSP que teve seu tempo de vida (duração) estourado.

A outra possibilidade de ocorrência do Evento 3, é quando há necessidade de preempção de LSPs estabelecidas. As seguintes atualizações na entidade gerenciadora após o desestabelecimento de um LSP devido à necessidade de preempção de LSPs:

- Retirada dos LSPs preemptados da base de dados de LSPs estabelecidos agrupados por CT, em todos os enlaces às quais os mesmos estiverem associados;
- Atualização da variável A_BC[n] (banda atual ocupada pelo determinada restrição de banda) nos enlaces do caminho que tiveram LSP(s) preemptado(s).

E por fim outra possibilidade de ocorrência do Evento 3, é quando há necessidade de devolução de empréstimos de LSPs estabelecidas em conformidade com o modelo AllocCT-Sharing. As seguintes atualizações na entidade gerenciadora após o desestabelecimento de um LSP devido à necessidade de devolução de empréstimos de LSPs:

- Retirada dos LSPs devolvidos da base de dados de LSPs estabelecidos agrupados por CT, em todos os enlaces às quais os mesmos estiverem associados;
- Atualização da variável A_BC[n] (banda atual ocupada pelo determinada restrição de banda) nos enlaces do caminho que tiveram LSP(s) devolvidos(s).

ANEXO B – Código das principais funções utilizadas para simulação

```
/********** ESTRUTURAS E DEFINIÇÕES **********/
#define MaxCaminhos 10
#define MaxOverlapFactor 5
#define MaxSaltos 5
#define NumEnlacesFullDuplex 12
#define MaxH 50
#define MaxClassType 3
#define LINKS 6
# define MAX_SIMULATIONS          1
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include<conio.h>
#include <cfloat>
typedef double real;
struct lsp {
        double Carga;
        int H;
        int src;
        int dest;
        int NumParOD;
        int PrioridadeOD;
        int NumCaminhosOD;
        int NumHops;
        char MatCaminhos[MaxCaminhos +1][MaxSaltos +1];
        char CaminhoEscolhido [MaxSaltos +1];
        int CT;
        int BC;
        int preeemptadaPorDebito;
};

struct no {
        struct lsp item;
        struct no *prox;
};

struct lista {
        struct no *primeiro;
        struct no *ultimo;
};

struct BC_RDM
{
        struct lista LPreemptebla[MaxH +1];
};

struct link
{
        double CargaEnlace;
        double CargaEnlaceAtual;
```

```
        double CargaEnlaceAnterior;
        double CargaResidual;
        double BandaPreemptada;
        double TotalBandaPreemptada;
        double BandaPreemptadaCT[MaxClassType];
        char DescEnlace;
        int NumEnlace;
        int CustoEnlace;
        int nosrc;
        int nodest;
        struct BC_RDM CT[MaxClassType];
        struct lista LLSPs;
        struct lista LPreempted;
        double BCPos[MaxClassType];
        double BCAtual[MaxClassType];
        double BCAcumulado[MaxClassType];
        double BCAcumuladoAUX[MaxClassType];
} lnk [LINKS+1];

struct simul
{
        char name [50];
        struct evchain *evc_begin;
        struct evchain *evc_end;
} sim [MAX_SIMULATIONS];

struct evchain
{
        double ev_time;
        int ev_tkn;
        int ev_type;
        struct evchain *ev_prior;
        struct evchain *ev_next;
        struct no *ev_tkn_p;
};

/********** VARIÁVEIS GLOBAIS **********/

char arquivo_saida[20];
struct lista LPreempted;
struct lista LBlocked;
struct lista LAdapted;
int Total_LSPs_Preempted_Debt=0;
int Total_LSPs_Preempted=0;
int Total_LSPs_Blocked=0;
int N_LSPs_Preempted[MaxClassType];
int N_LSPs_Preempted_Debt[MaxClassType];
int N_LSPs_Blocked[MaxClassType];
int N_LSPs_Adapted = 0;
double TotalBandaReduct = 0;
double TotalBandaPreempted = 0;
double TotalBandaAtendida = 0;
double TotalBandaAdapted = 0;
int VetorLSPsPrempatadas[MaxH +1];
double BC[MaxClassType];
```

```
int contador=0;


//Parâmetros da Cadeia de Eventos
char arquivo_saida[20];
double clock=0;
int event;
int sn = 0;
int LSPs_Geradas=0;
int LSP_Number=0;
int LSP_Number2=0;

FILE *fp;
FILE *fpXML;

/********** CRIACAO DA LSP E ALOCACAO DE AREA DE MEMÓRIA **********/

struct no *create_LSP (void)
{
	struct no *LSP
	LSP = malloc(sizeof(struct no));
	if (LSP == NULL)
	{
		printf ("\n Erro - fc creat packet - nao tem mais memoria para alocar para pkt");
		exit (0);
	}
	LSP->item.preeemptadaPorDebito=0;
	return (LSP);
}

/****************** PROGRAMA PRINCIPAL ********************/

main()
{
	int Establish;
	int AllocCT=0;
	struct lista CT0, CT1, CT2, L1ordem;
	struct lista LNovaReq; // Lista que contém os LSPs que foram preemptados
	double ie_t=0.0, ia_t=0.0;
	struct no *dados;
	int i,j,n_lnk=0;
	int nEnl;
	int countXML=1;
	double TotalBandaPreemptada = 0;
	if(AllocCT==1)
		sprintf(arquivo_saida, "AllocCT-Sharing.txt");
	else
		sprintf(arquivo_saida, "rdm.txt"); /
	fp = fopen(arquivo_saida,"w");
	if(AllocCT==1)
		fpXML = fopen("AllocCT-Sharing.xml","w");
	else
		fpXML = fopen("rdm.xml","w");
```

```
/***** INICIALIZAÇÃO LISTAS PARÂMETROS DA MATRIZ DE TRÁFEGO POR CT ********/
        inicia_lista(&CT0);
        inicia_lista(&CT1);
        inicia_lista(&CT2);
        inicia_lista(&L1ordem);
        inicia_lista(&LNovaReq);
        inicia_lista(&LPreempted);
        inicia_lista(&LAdapted);
        inicia_lista( &LBlocked);
        Init_evchain();
        for (j=1; j<= LINKS; j++)
                inicia_lista(&lnk[j].LLSPs);
        for (j=1; j<= MaxH; j++)
                VetorLSPsPrempatadas[j] =0;
        for (nEnl=1; nEnl<= LINKS; nEnl++)
        {
                for (j=0; j< MaxClassType; j++)
                        lnk[nEnl].BCAcumuladoAUX[j];
        }

        // Inicializando o modelo de restrição de banda RDM
        BC[2] = 40; // CT2
        BC[1] = 70; // CT1 + CT2
        BC[0] = 100;

        /*************** CONFIGURAÇÃO DOS ENLACES *****************/

        //  E1 Enlace A (1->2)
        lnk[1].DescEnlace = 'A';
        lnk[1].NumEnlace = 1;
        lnk[1].CustoEnlace = 1;
        lnk[1].CargaEnlace = 622;
        lnk[1].CargaEnlaceAtual = 0;
        lnk[1].CargaEnlaceAnterior = 0;
        lnk[1].CargaResidual = 0;
        lnk[1].TotalBandaPreemptada = 0;
        lnk[1].BandaPreemptada = 0;
        lnk[1].nosrc = 1;
        lnk[1].nodest = 2;
        n_lnk++;

        //  E2 Enlace B (2->3)
        lnk[2].DescEnlace = 'B';
        lnk[2].NumEnlace = 2;
        lnk[2].CustoEnlace = 1;
        lnk[2].CargaEnlace = 622;
        lnk[2].CargaEnlaceAtual = 0;
        lnk[2].CargaEnlaceAnterior = 0;
        lnk[2].CargaResidual = 0;
        lnk[2].TotalBandaPreemptada = 0;
        lnk[2].BandaPreemptada = 0;
        lnk[2].nosrc = 2;
        lnk[2].nodest = 3;
        n_lnk++;
```

```
//E3 Enlace C (1->3)
lnk[3].DescEnlace = 'C';
lnk[3].NumEnlace = 3;
lnk[3].CustoEnlace = 1;
lnk[3].CargaEnlace = 622;
lnk[3].CargaEnlaceAtual = 0;
lnk[3].CargaEnlaceAnterior = 0;
lnk[3].CargaResidual = 0;
lnk[3].TotalBandaPreemptada = 0;
lnk[3].BandaPreemptada = 0;
lnk[3].nosrc = 1;
lnk[3].nodest = 3;
n_lnk++;

// E4 Enlace D (1->4)
lnk[4].DescEnlace = 'D';
lnk[4].NumEnlace = 4;
lnk[4].CustoEnlace = 1;
lnk[4].CargaEnlace = 622;
lnk[4].CargaEnlaceAtual = 0;
lnk[4].CargaEnlaceAnterior = 0;
lnk[4].CargaResidual = 0;
lnk[4].TotalBandaPreemptada = 0;
lnk[4].BandaPreemptada = 0;
lnk[4].nosrc = 1;
lnk[4].nodest = 4;
n_lnk++;

// E5 Enlace E (4->5)
lnk[5].DescEnlace = 'E';
lnk[5].NumEnlace = 5;
lnk[5].CustoEnlace = 1;
lnk[5].CargaEnlace = 622;
lnk[5].CargaEnlaceAtual = 0;
lnk[5].CargaEnlaceAnterior = 0;
lnk[5].CargaResidual = 0;
lnk[5].TotalBandaPreemptada = 0;
lnk[5].BandaPreemptada = 0;
lnk[5].nosrc = 4;
lnk[5].nodest = 5;
n_lnk++;

// E6 Enlace F (5->3)
lnk[6].DescEnlace = 'F';
lnk[6].NumEnlace = 6;
lnk[6].CustoEnlace = 1;
lnk[6].CargaEnlace = 622;
lnk[6].CargaEnlaceAtual = 0;
lnk[6].CargaEnlaceAnterior = 0;
lnk[6].CargaResidual = 0;
lnk[6].TotalBandaPreemptada = 0;
lnk[6].BandaPreemptada = 0;
lnk[6].nosrc = 5;
lnk[6].nodest = 3;
```

```
n_lnk++;

/************** INICIALIZACAO DAS LSPs (1->3) ******************/

/***** LSP associada ao CT0 *****/
LSP_Number++;
dados = create_LSP();
dados->item.CargaReduzida = 0;
dados->item.src = 1;
dados->item.dest = 3;
dados->item.NumCaminhosOD = 3;
dados->item.NumParOD = LSP_Number;
dados->item.PrioridadeOD = 0;
dados->item.H = 8 - dados->item.PrioridadeOD;
dados->item.NumHops = 0;
dados->item.CT = 0;
dados->item.Carga = 0;

// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
{
        for (j=1; j<= MaxSaltos; j++)
        {
                dados->item.CaminhoEscolhido[j] = 0;
                dados->item.MatCaminhos[i][j] = 0;
        }
}

// DEFINIÇÃO DA MATRIZ DE CAMINHO
dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
dados->item.MatCaminhos[2][2] = 'B';

dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
dados->item.MatCaminhos[3][2] = 'E';
dados->item.MatCaminhos[3][3] = 'F';

schedulep (1, 500.0, LSP_Number, dados);

/***** LSP associada ao CT1 *****/
LSP_Number++;
dados = create_LSP();
dados->item.CargaReduzida = 0;
dados->item.src = 1;
dados->item.dest = 3;
dados->item.NumCaminhosOD = 3;
dados->item.NumParOD = LSP_Number;
dados->item.PrioridadeOD = 0;
dados->item.H = 8 - dados->item.PrioridadeOD;
dados->item.NumHops = 0;
dados->item.CT = 1;
dados->item.Carga = 0;
```

```
// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
{
	for (j=1; j<= MaxSaltos; j++)
	{
		dados->item.CaminhoEscolhido[j] = 0;
		dados->item.MatCaminhos[i][j] = 0;
	}
}
// DEFINIÇÃO DA MATRIZ DE CAMINHO
dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
dados->item.MatCaminhos[2][2] = 'B';

dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
dados->item.MatCaminhos[3][2] = 'E';
dados->item.MatCaminhos[3][3] = 'F';

schedulep (1, 300.0, LSP_Number, dados);

/***** LSP associada ao CT2 *****/
LSP_Number++;
dados = create_LSP();
dados->item.CargaReduzida = 0;
dados->item.src = 1;
dados->item.dest = 3;
dados->item.NumCaminhosOD = 3;
dados->item.NumParOD = LSP_Number;
dados->item.PrioridadeOD = 0;
dados->item.H = 8 - dados->item.PrioridadeOD;
dados->item.NumHops = 0;
dados->item.CT = 2;
dados->item.Carga = 0;

// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
{
	for (j=1; j<= MaxSaltos; j++)
	{
		dados->item.CaminhoEscolhido[j] = 0;
		dados->item.MatCaminhos[i][j] = 0;
	}
}

// DEFINIÇÃO DA MATRIZ DE CAMINHO
dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
dados->item.MatCaminhos[2][2] = 'B';

dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
dados->item.MatCaminhos[3][2] = 'E';
dados->item.MatCaminhos[3][3] = 'F';
```

```
	schedulep (1, 0.0, LSP_Number, dados);

/************** INICIALIZACAO DAS LSPs (2->3) ********************/

	/***** LSP associada ao CT0 *****/
	LSP_Number++;
	dados = create_LSP();
	dados->item.CargaReduzida = 0;
	dados->item.src = 2;
	dados->item.dest = 3;
	dados->item.NumCaminhosOD = 3;
	dados->item.NumParOD = LSP_Number;
	dados->item.PrioridadeOD = 0;
	dados->item.H = 8 - dados->item.PrioridadeOD;
	dados->item.NumHops = 0;
	dados->item.CT = 0;
	dados->item.Carga = 0;

	// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
	for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
	{
		for (j=1; j<= MaxSaltos; j++)
		{
			dados->item.CaminhoEscolhido[j] = 0;
			dados->item.MatCaminhos[i][j] = 0;
		}
	}
	// DEFINIÇÃO DA MATRIZ DE CAMINHO
	dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1

	schedulep (4, 500.0, LSP_Number, dados);

	/***** LSP associada ao CT1 *****/
	LSP_Number++;
	dados = create_LSP();
	dados->item.CargaReduzida = 0;
	dados->item.src = 2;
	dados->item.dest = 3;
	dados->item.NumCaminhosOD = 3;
	dados->item.NumParOD = LSP_Number;
	dados->item.PrioridadeOD = 0;
	dados->item.H = 8 - dados->item.PrioridadeOD;
	dados->item.NumHops = 0;
	dados->item.CT = 1;
	dados->item.Carga = 0;

	// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
	for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
	{
		for (j=1; j<= MaxSaltos; j++)
		{
			dados->item.CaminhoEscolhido[j] = 0;
			dados->item.MatCaminhos[i][j] = 0;
		}
	}
```

```
		// DEFINIÇÃO DA MATRIZ DE CAMINHO
		dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1

		schedulep (4, 300.0, LSP_Number, dados);

		/***** LSP associada ao CT2 *****/
		LSP_Number++;
		dados = create_LSP();
		dados->item.CargaReduzida = 0;
		dados->item.src = 2;
		dados->item.dest = 3;
		dados->item.NumCaminhosOD = 3;
		dados->item.NumParOD = LSP_Number;
		dados->item.PrioridadeOD = 0;
		dados->item.H = 8 - dados->item.PrioridadeOD;
		dados->item.NumHops = 0;
		dados->item.CT = 2;
		dados->item.Carga = 0;

		// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
		for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
		{
			for (j=1; j<= MaxSaltos; j++)
			{
				dados->item.CaminhoEscolhido[j] = 0;
				dados->item.MatCaminhos[i][j] = 0;
			}
		}

		// DEFINIÇÃO DA MATRIZ DE CAMINHO
		dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1

		schedulep (4, 0.0, LSP_Number, dados);

/************** INICIALIZACAO DAS LSPs (4->3) ********************/

/***** LSP associada ao CT0 *****/
LSP_Number++;
dados = create_LSP();
dados->item.CargaReduzida = 0;
dados->item.src = 4;
dados->item.dest = 3;
dados->item.NumCaminhosOD = 3;
dados->item.NumParOD = LSP_Number;
dados->item.PrioridadeOD = 0;
dados->item.H = 8 - dados->item.PrioridadeOD;
dados->item.NumHops = 0;
dados->item.CT = 0;
dados->item.Carga = 0;

// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
{
	for (j=1; j<= MaxSaltos; j++)
```

```
	{
		dados->item.CaminhoEscolhido[j] = 0;
		dados->item.MatCaminhos[i][j] = 0;
	}
}

// DEFINIÇÃO DA MATRIZ DE CAMINHO
dados->item.MatCaminhos[1][1] = 'E'; // Caminho 3
dados->item.MatCaminhos[1][2] = 'F';

schedulep (5, 500.0, LSP_Number, dados);


/***** LSP associada ao CT1 *****/
LSP_Number++;
dados = create_LSP();
dados->item.CargaReduzida = 0;
dados->item.src = 4;
dados->item.dest = 3;
dados->item.NumCaminhosOD = 3;
dados->item.NumParOD = LSP_Number;
dados->item.PrioridadeOD = 0;
dados->item.H = 8 - dados->item.PrioridadeOD;
dados->item.NumHops = 0;
dados->item.CT = 1;
dados->item.Carga = 0;

// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
{
	for (j=1; j<= MaxSaltos; j++)
	{
		dados->item.CaminhoEscolhido[j] = 0;
		dados->item.MatCaminhos[i][j] = 0;
	}
}

// DEFINIÇÃO DA MATRIZ DE CAMINHO
dados->item.MatCaminhos[1][1] = 'E'; // Caminho 3
dados->item.MatCaminhos[1][2] = 'F';

schedulep (5, 300.0, LSP_Number, dados);

/***** LSP associada ao CT2 *****/
LSP_Number++;
dados = create_LSP();
dados->item.CargaReduzida = 0;
dados->item.src = 4;
dados->item.dest = 3;
dados->item.NumCaminhosOD = 3;
dados->item.NumParOD = LSP_Number;
dados->item.PrioridadeOD = 0;
dados->item.H = 8 - dados->item.PrioridadeOD;
dados->item.NumHops = 0;
dados->item.CT = 2;
```

```
dados->item.Carga = 0;

// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
{
	for (j=1; j<= MaxSaltos; j++)
	{
		dados->item.CaminhoEscolhido[j] = 0;
		dados->item.MatCaminhos[i][j] = 0;
	}
}

// DEFINIÇÃO DA MATRIZ DE CAMINHO
dados->item.MatCaminhos[1][1] = 'E'; // Caminho 3
dados->item.MatCaminhos[1][2] = 'F';

schedulep (5, 0.0, LSP_Number, dados);

/***************** SIMULAÇÃO*******************/

srand(321);

//while(simtime()<20)

fprintf(fpXML,"<?xml version='1.0'?>\r\n");
fprintf(fpXML,"<simulacao>\n");

while(LSPs_Geradas<1000)
{
	imprime_evchain();
	fprintf(fp,"\n\n");
	dados = causep (&event, &LSP_Number2);
	switch(event)
	{

	case 1:

	LSPs_Geradas++;
	schedulep (2, 0.0, dados->item.NumParOD, dados);
	LSP_Number++;

	if(dados->item.CT == 0)
	{
		ie_t = expntl(4);
		/***** LSP associada ao CT0 *****/
		dados = create_LSP();
		dados->item.CargaReduzida = 0;
		dados->item.src = 1;
		dados->item.dest = 3;
		dados->item.NumCaminhosOD = 3;
		dados->item.NumParOD = LSP_Number;
		dados->item.PrioridadeOD = (int)uniform(0,3);
		dados->item.H = 8 - dados->item.PrioridadeOD;
		dados->item.NumHops = 0;
		dados->item.CT = 0;
```

```
        dados->item.Carga = (int)uniform(5,20);

        // INICIALIZAÇÃO DA MATRIZ DE CAMINHO
        for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
        {
                for (j=1; j<= MaxSaltos; j++)
                {
                        dados->item.CaminhoEscolhido[j] = 0;
                        dados->item.MatCaminhos[i][j] = 0;
                }
        }

        // DEFINIÇÃO DA MATRIZ DE CAMINHO
        if(dados->item.src == 1)
        {
                dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

                dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
                dados->item.MatCaminhos[2][2] = 'B';

                dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
                dados->item.MatCaminhos[3][2] = 'E';
                dados->item.MatCaminhos[3][3] = 'F';
        }
        if(dados->item.src == 2)
                dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1

        if(dados->item.src == 4)
        {
                dados->item.MatCaminhos[1][1] = 'E'; // Caminho 1
                dados->item.MatCaminhos[1][2] = 'F';
        }
        schedulep (1, ie_t, dados->item.NumParOD, dados);
        break;
}

if(dados->item.CT == 1)
{
        ie_t = expntl(2);
        /***** LSP associada ao CT1 *****/
        dados = create_LSP();
        dados->item.CargaReduzida = 0;
        dados->item.src = 1;
        dados->item.dest = 3;
        dados->item.NumCaminhosOD = 3;
        dados->item.NumParOD = LSP_Number;
        dados->item.PrioridadeOD = (int)uniform(0,3);
        dados->item.H = 8 - dados->item.PrioridadeOD;
        dados->item.NumHops = 0;
        dados->item.CT = 1;
        dados->item.Carga = (int)uniform(5,20);

        // INICIALIZAÇÃO DA MATRIZ DE CAMINHO
        for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
        {
```

```
                for (j=1; j<= MaxSaltos; j++)
                {
                        dados->item.CaminhoEscolhido[j] = 0;
                        dados->item.MatCaminhos[i][j] = 0;
                }
        }

        // DEFINIÇÃO DA MATRIZ DE CAMINHO
        if(dados->item.src == 1)
        {
                dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

                dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
                dados->item.MatCaminhos[2][2] = 'B';

                dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
                dados->item.MatCaminhos[3][2] = 'E';
                dados->item.MatCaminhos[3][3] = 'F';
        }

        if(dados->item.src == 2)
                dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1

        if(dados->item.src == 4)
        {
                dados->item.MatCaminhos[1][1] = 'E'; // Caminho 1
                dados->item.MatCaminhos[1][2] = 'F';
        }

        schedulep (1, ie_t, dados->item.NumParOD, dados);
        break;

}

if(dados->item.CT == 2)
{
        ie_t = expntl(1);
        /***** LSP associada ao CT2 *****/
        dados = create_LSP();
        dados->item.CargaReduzida = 0;
        dados->item.src = 1;
        dados->item.dest = 3;
        dados->item.NumCaminhosOD = 3;
        dados->item.NumParOD = LSP_Number;
        dados->item.PrioridadeOD = (int)uniform(0,3);
        dados->item.H = 8 - dados->item.PrioridadeOD;
        dados->item.NumHops = 0;
        dados->item.CT = 2;
        dados->item.Carga = (int)uniform(5,20);

        // INICIALIZAÇÃO DA MATRIZ DE CAMINHO
        for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
        {
                for (j=1; j<= MaxSaltos; j++)
                {
```

```
                        dados->item.CaminhoEscolhido[j] = 0;
                        dados->item.MatCaminhos[i][j] = 0;
                }
        }

        // DEFINIÇÃO DA MATRIZ DE CAMINHO
        if(dados->item.src == 1)
        {
                dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

                dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
                dados->item.MatCaminhos[2][2] = 'B';

                dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
                dados->item.MatCaminhos[3][2] = 'E';
                dados->item.MatCaminhos[3][3] = 'F';
        }

        if(dados->item.src == 2)
                dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1

        if(dados->item.src == 4)
        {
                dados->item.MatCaminhos[1][1] = 'E'; // Caminho 1
                dados->item.MatCaminhos[1][2] = 'F';
        }
        schedulep (1, ie_t, dados->item.NumParOD, dados);
        break;
}
else
{
        printf("CT %d nao definida na simulacao!\n", dados->item.CT);
        break;
}

case 2:
if(AllocCT==1)
        Establish = TryPath_CSPF_WithAllocCTSharing(dados,100);
else
        Establish = TryPath_CSPF_WithPreemption(dados);

//Outros modelos
//Establish = TryPath_MinDSTEPreemptWithDebtAndReduct(dados,100);
//Establish = TryPath_CSPF_NoPreemption(dados);
//Establish = TryPath_CSPF_WithDebtPreemption(dados);
//Establish = TryPath_MinDSTEPreemptWithDebt(dados);
//Establish =TryPath_MinDSTEPreempt_v3(dados);
//Establish =TryPath_MinDSTEPreempt_v4(dados);
if(Establish==1)
{
        if(dados->item.CT==0)
                ia_t = expntl(150);
        if(dados->item.CT==1)
                ia_t = expntl(150);
        if(dados->item.CT==2)
```

```
                ia_t = expntl(150);
            schedulep (3, ia_t,dados->item.NumParOD, dados);
        }
        else
        {
            fprintf(fp,"LSP%d(%d-%d)(CT=%d): %.2lf - Bloqueada\n",dados->item.NumParOD,dados->item.src, dados->item.dest,dados->item.CT,dados->item.Carga);
        }
        //Estatística XML
        //fprintf(fpXML,"<id>%d",LSP_Number);
        //fprintf(fpXML,"</id>\n");
        fprintf(fpXML,"<interacao>\n");
        fprintf(fpXML,"<interacao_total_lsps_geradas>%d</interacao_total_lsps_geradas>\n", LSPs_Geradas);
        for(j=0;j<MaxClassType;j++)
            fprintf(fpXML,"<interacao_lsps_preempted_class_%d>%d</interacao_lsps_preempted_class_%d>\n", j, N_LSPs_Preempted[j],j);
        for(j=0;j<MaxClassType;j++)
            fprintf(fpXML,"<interacao_lsps_preempted_class_debt_%d>%d</interacao_lsps_preempted_class_debt_%d>\n", j, N_LSPs_Preempted_Debt[j],j);
        for(j=0;j<MaxClassType;j++)
            fprintf(fpXML,"<interacao_lsps_blocked_class_%d>%d</interacao_lsps_blocked_class_%d>\n", j, N_LSPs_Blocked[j],j);

        fprintf(fpXML,"<interacao_total_lsps_preemptadas>%d</interacao_total_lsps_preemptadas>\n", Total_LSPs_Preempted);

        fprintf(fpXML,"<interacao_total_lsps_bloqueadas>%d</interacao_total_lsps_bloqueadas>\n", Total_LSPs_Blocked);

        fprintf(fpXML,"<interacao_total_lsps_preemptadas_debt>%d</interacao_total_lsps_preemptadas_debt>\n", Total_LSPs_Preempted_Debt);

        fprintf(fpXML,"<interacao_total_banda_atendida>%.2lf</interacao_total_banda_atendida>\n", TotalBandaAtendida);

        for(i=1;i<=LINKS;i++)
        {
            fprintf(fpXML,"<enlace>\n");
            fprintf(fpXML,"<enlace_numero>%d",i);
            fprintf(fpXML,"</enlace_numero>\n");

            fprintf(fpXML,"<enlace_carga>%.2lf", lnk[i].CargaEnlaceAtual);
            fprintf(fpXML,"</enlace_carga>\n");
            fprintf(fpXML,"<enlace_residual>%.2lf", lnk[i].CargaResidual);
            fprintf(fpXML,"</enlace_residual>\n");
            for(j=0;j<MaxClassType;j++)
            {
                if(j!=MaxClassType-1)
                    fprintf(fpXML,"<enlace_carga_ct_%d>%lf</enlace_carga_ct_%d>\n", j, lnk[i].BCAcumulado[j]-lnk[i].BCAcumulado[j+1], j);
                else
                    fprintf(fpXML,"<enlace_carga_ct_%d>%lf</enlace_carga_ct_%d>\n", j, lnk[i].BCAcumulado[j], j);
            }
```

```
		fprintf(fpXML,"</enlace>\n");
}
fprintf(fpXML,"</interacao>\n");
break;

case 3:
//Incrementa o total de banda atendida
TotalBandaAtendida+=dados->item.Carga;
RetiraLSPDesestablish(LSP_Number2, dados);
Atualiza_BCs_porEnlace_posDesestabelecimento(dados);
Atualiza_CargadosEnlaces();
imprime_BCAcumulado();
break;

case 4:

/************** INICIALIZACAO DAS LSPs (2->3) ******************/
LSPs_Geradas++;
schedulep (2, 0.0, dados->item.NumParOD, dados);
LSP_Number++;

if(dados->item.CT == 0)
{
		ie_t = expntl(20);
		/***** LSP associada ao CT0 *****/
		dados = create_LSP();
		dados->item.CargaReduzida = 0;
		dados->item.src = 2;
		dados->item.dest = 3;
		dados->item.NumCaminhosOD = 3;
		dados->item.NumParOD = LSP_Number;
		dados->item.PrioridadeOD = (int)uniform(0,3);
		dados->item.H = 8 - dados->item.PrioridadeOD;
		dados->item.NumHops = 0;
		dados->item.CT = 0;
		dados->item.Carga = (int)uniform(5,20);

		// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
		for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
		{
			for (j=1; j<= MaxSaltos; j++)
			{
				dados->item.CaminhoEscolhido[j] = 0;
				dados->item.MatCaminhos[i][j] = 0;
			}
		}
		// DEFINIÇÃO DA MATRIZ DE CAMINHO
		if(dados->item.src == 1)
		{
			dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

			dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
			dados->item.MatCaminhos[2][2] = 'B';

			dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
```

```
                dados->item.MatCaminhos[3][2] = 'E';
                dados->item.MatCaminhos[3][3] = 'F';
        }

        if(dados->item.src == 2)
                dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1

        if(dados->item.src == 4)
        {
                dados->item.MatCaminhos[1][1] = 'E'; // Caminho 1
                dados->item.MatCaminhos[1][2] = 'F';
        }
        schedulep (4, ie_t, dados->item.NumParOD, dados);
        break;
}
if(dados->item.CT == 1)
{
        ie_t = expntl(22);
        /***** LSP associada ao CT1 *****/
        dados = create_LSP();
        dados->item.CargaReduzida = 0;
        dados->item.src = 2;
        dados->item.dest = 3;
        dados->item.NumCaminhosOD = 3;
        dados->item.NumParOD = LSP_Number;
        dados->item.PrioridadeOD = (int)uniform(0,3);
        dados->item.H = 8 - dados->item.PrioridadeOD;
        dados->item.NumHops = 0;
        dados->item.CT = 1;
        dados->item.Carga = (int)uniform(5,20);

        // INICIALIZAÇÃO DA MATRIZ DE CAMINHO
        for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
        {
                for (j=1; j<= MaxSaltos; j++)
                {
                        dados->item.CaminhoEscolhido[j] = 0;
                        dados->item.MatCaminhos[i][j] = 0;
                }
        }
        // DEFINIÇÃO DA MATRIZ DE CAMINHO
        if(dados->item.src == 1)
        {
                dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

                dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
                dados->item.MatCaminhos[2][2] = 'B';

                dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
                dados->item.MatCaminhos[3][2] = 'E';
                dados->item.MatCaminhos[3][3] = 'F';
        }

        if(dados->item.src == 2)
                dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1
```

```
		if(dados->item.src == 4)
		{
			dados->item.MatCaminhos[1][1] = 'E'; // Caminho 1
			dados->item.MatCaminhos[1][2] = 'F';
		}
		schedulep (4, ie_t, dados->item.NumParOD, dados);
		break;
}

if(dados->item.CT == 2)
{
		ie_t = expntl(24);
		/***** LSP associada ao CT2 *****/
		dados = create_LSP();
		dados->item.CargaReduzida = 0;
		dados->item.src = 2;
		dados->item.dest = 3;
		dados->item.NumCaminhosOD = 3;
		dados->item.NumParOD = LSP_Number;
		dados->item.PrioridadeOD = (int)uniform(0,3);
		dados->item.H = 8 - dados->item.PrioridadeOD;
		dados->item.NumHops = 0;
		dados->item.CT = 2;
		dados->item.Carga = (int)uniform(5,20);

		// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
		for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
		{
			for (j=1; j<= MaxSaltos; j++)
			{
				dados->item.CaminhoEscolhido[j] = 0;
				dados->item.MatCaminhos[i][j] = 0;
			}
		}

		// DEFINIÇÃO DA MATRIZ DE CAMINHO
		if(dados->item.src == 1)
		{
			dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

			dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
			dados->item.MatCaminhos[2][2] = 'B';

			dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
			dados->item.MatCaminhos[3][2] = 'E';
			dados->item.MatCaminhos[3][3] = 'F';
		}

		if(dados->item.src == 2)
			dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1

		if(dados->item.src == 4)
		{
			dados->item.MatCaminhos[1][1] = 'E'; // Caminho 1
			dados->item.MatCaminhos[1][2] = 'F';
```

```
        }
        schedulep (4, ie_t, dados->item.NumParOD, dados);
        break;
}
case 5:
/************** INICIALIZACAO DAS LSPs (4->3) ******************/
LSPs_Geradas++;
schedulep (2, 0.0, dados->item.NumParOD, dados);
LSP_Number++;
if(dados->item.CT == 0)
{
        ie_t = expntl(20);
        /***** LSP associada ao CT0 *****/
        dados = create_LSP();
        dados->item.CargaReduzida = 0;
        dados->item.src = 4;
        dados->item.dest = 3;
        dados->item.NumCaminhosOD = 3;
        dados->item.NumParOD = LSP_Number;
        dados->item.PrioridadeOD = (int)uniform(0,3);
        dados->item.H = 8 - dados->item.PrioridadeOD;
        dados->item.NumHops = 0;
        dados->item.CT = 0;
        dados->item.Carga = (int)uniform(5,20);

        // INICIALIZAÇÃO DA MATRIZ DE CAMINHO
        for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
        {
                for (j=1; j<= MaxSaltos; j++)
                {
                        dados->item.CaminhoEscolhido[j] = 0;
                        dados->item.MatCaminhos[i][j] = 0;
                }
        }

        // DEFINIÇÃO DA MATRIZ DE CAMINHO
        if(dados->item.src == 1)
        {
                dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

                dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
                dados->item.MatCaminhos[2][2] = 'B';

                dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
                dados->item.MatCaminhos[3][2] = 'E';
                dados->item.MatCaminhos[3][3] = 'F';
        }
        if(dados->item.src == 2)
                dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1
        if(dados->item.src == 4)
        {
                dados->item.MatCaminhos[1][1] = 'E'; // Caminho 1
                dados->item.MatCaminhos[1][2] = 'F';
        }
        schedulep (5, ie_t, dados->item.NumParOD, dados);
```

```
		break;
	}
	if(dados->item.CT == 1)
	{
		ie_t = expntl(22);
		/***** LSP associada ao CT1 *****/
		dados = create_LSP();
		dados->item.CargaReduzida = 0;
		dados->item.src = 4;
		dados->item.dest = 3;
		dados->item.NumCaminhosOD = 3;
		dados->item.NumParOD = LSP_Number;
		dados->item.PrioridadeOD = (int)uniform(0,3);
		dados->item.H = 8 - dados->item.PrioridadeOD;
		dados->item.NumHops = 0;
		dados->item.CT = 1;
		dados->item.Carga = (int)uniform(5,20);
		// INICIALIZAÇÃO DA MATRIZ DE CAMINHO
		for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
		{
			for (j=1; j<= MaxSaltos; j++)
			{
				dados->item.CaminhoEscolhido[j] = 0;
				dados->item.MatCaminhos[i][j] = 0;
			}
		}
		// DEFINIÇÃO DA MATRIZ DE CAMINHO
		if(dados->item.src == 1)
		{
			dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

			dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
			dados->item.MatCaminhos[2][2] = 'B';

			dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
			dados->item.MatCaminhos[3][2] = 'E';
			dados->item.MatCaminhos[3][3] = 'F';
		}
		if(dados->item.src == 2)
			dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1
		if(dados->item.src == 4)
		{
			dados->item.MatCaminhos[1][1] = 'E'; // Caminho 1
			dados->item.MatCaminhos[1][2] = 'F';
		}
		schedulep (5, ie_t, dados->item.NumParOD, dados);
		break;
	}
	if(dados->item.CT == 2)
	{
		ie_t = expntl(24);
		/***** LSP associada ao CT2 *****/
		dados = create_LSP();
		dados->item.CargaReduzida = 0;
		dados->item.src = 4;
```

```
                dados->item.dest = 3;
                dados->item.NumCaminhosOD = 3;
                dados->item.NumParOD = LSP_Number;
                dados->item.PrioridadeOD = (int)uniform(0,3);
                dados->item.H = 8 - dados->item.PrioridadeOD;
                dados->item.NumHops = 0;
                dados->item.CT = 2;
                dados->item.Carga = (int)uniform(5,20);
                // INICIALIZAÇÃO DA MATRIZ DE CAMINHO
                for (i=1; i<= MaxCaminhos; i++) // Inicializa a Matriz de Roteamento
                {
                        for (j=1; j<= MaxSaltos; j++)
                        {
                                dados->item.CaminhoEscolhido[j] = 0;
                                dados->item.MatCaminhos[i][j] = 0;
                        }
                }
                // DEFINIÇÃO DA MATRIZ DE CAMINHO
                if(dados->item.src == 1)
                {
                        dados->item.MatCaminhos[1][1] = 'C'; // Caminho 1

                        dados->item.MatCaminhos[2][1] = 'A'; // Caminho 2
                        dados->item.MatCaminhos[2][2] = 'B';

                        dados->item.MatCaminhos[3][1] = 'D'; // Caminho 3
                        dados->item.MatCaminhos[3][2] = 'E';
                        dados->item.MatCaminhos[3][3] = 'F';
                }
                if(dados->item.src == 2)
                        dados->item.MatCaminhos[1][1] = 'B'; // Caminho 1
                if(dados->item.src == 4)
                {
                        dados->item.MatCaminhos[1][1] = 'E'; // Caminho 1
                        dados->item.MatCaminhos[1][2] = 'F';
                }
                schedulep (5, ie_t, dados->item.NumParOD, dados);
                break;
        }
        else
        {
                printf("CT %d nao definida na simulacao!\n", dados->item.CT);
                break;
        }

}
}

fprintf(fp,"\n");
fprintf(fp, "---------- Carga dos Enlaces Após Dist Inicial ----------\n\n");
for(i=1;i<=LINKS;i++)
{
        fprintf(fp,"Carga_Enlace%d = %.2lf \n", i, lnk[i].CargaEnlaceAtual);
        fprintf(fp,"Carga_Residual_Enlace%d = %.2lf \n\n", i, lnk[i].CargaResidual);
}
```

```
fprintf(fp, "-------------------------------------------------------------\n");


/*** BANDA PREEMPTADA EM CADA ENLACE APÓS PRIMEIRA PREEMPÇÃO *********/
fprintf(fp,"\n");
fprintf(fp, "------------------ Listas com LSPs Preemptáveis (Após Execução da Preempção)  ----------
\n\n");
for(i=1; i<=LINKS;i++)
{
	for (j=0;j<MaxClassType;j++)
		lnk[i].BandaPreemptada = lnk[i].BandaPreemptada + lnk[i].BandaPreemptadaCT[j];
	//fprintf(fp,"Banda_Preemptada_Enlace%d = %.3lf\n", i, lnk[i].BandaPreemptada);

}
for(i=0;i<MaxClassType;i++)
	fprintf(fp,"LSPs_Preempted_Class[%d] = %d\n", i, N_LSPs_Preempted[i]);
fprintf(fp, "\n");
for(i=0;i<MaxClassType;i++)
	fprintf(fp,"LSPs_Preempted_Class_Debt[%d] = %d\n", i, N_LSPs_Preempted_Debt[i]);
fprintf(fp, "\n");
for(i=0;i<MaxClassType;i++)
	fprintf(fp,"LSPs_Blocked_Class[%d] = %d\n", i, N_LSPs_Blocked[i]);
fprintf(fp, "\n");
fprintf(fp,"Número de LSPs Preemptadas = %d\n", Total_LSPs_Preempted);
fprintf(fp,"Número de LSPs Preemptadas Debt = %d\n", Total_LSPs_Preempted_Debt);
fprintf(fp,"Número de LSPs Bloqueadas = %d\n", Total_LSPs_Blocked);
fprintf(fp,"Número de LSPs Geradas = %d\n", LSPs_Geradas);
fprintf(fp,"Total de Banda Atendida = %.2lf\n", TotalBandaAtendida);
fprintf(fp, "\n-------------------------------------------------------------\n");
/**** REIMPRESSÃO LISTAS ATUAIS DE LSPs PREEMPTÁVEIS APÓS 1ª RODADA DE
PREEMPÇÃO ******/

//imprime_LSPsPreemptaveispoCTporEnlace();

/******* IMPRESSÃO FINAL DOS BCS *********/
//imprime_BCAcumuladoAUX_Atual();
//imprime_BCAcumulado_Atual();
/**** REIMPRESSÃO DOS VALORES DOS BCS ATUAIS ******/
/*
	fprintf(fp, "---------------- Valor final dos BCs ----------\n\n");
	for(nEnl=1;nEnl<=LINKS;nEnl++)
	{
		fprintf(fp,"------- Enlace%d -------\n", lnk[nEnl].NumEnlace);
		for(i=0;i<MaxClassType;i++)
		{
			//fprintf(fp,"BandaCT[%d] = %lf\n", i, lnk[nEnl].BCPos[i]);
			fprintf(fp,"BandaCT[%d] = %lf\n", i, lnk[nEnl].BCAcumuladoAUX[i]);
		}
		fprintf(fp,"\n");
	}
	fprintf(fp, "\n-------------------------------------------------------------\n");
*/

/********** IMPRESSÃO TRÁFEGO DOS ENLACES APÓS PREEMPÇÃO **********/
fprintf(fp,"\n");
```

```
fprintf(fp, "---------- Carga dos Enlaces Após Dist Inicial ----------\n\n");
for(i=1;i<=LINKS;i++)
{
	fprintf(fp,"Carga_Enlace%d = %.2lf \n", i, lnk[i].CargaEnlaceAtual);
	fprintf(fp,"Carga_Residual_Enlace%d = %.2lf \n\n", i, lnk[i].CargaResidual);
}
fprintf(fp, "-----------------------------------------------------------\n");
fprintf(fpXML,"</simulacao>\n");

fclose(fpXML);
fclose(fp);
}

/*** FUNÇÕES AUXILIARES DOS ALGORITMOS DE ROTEAMENTO E PREEMPÇÃO ***/

void inicia_lista(struct lista *L)
{
	L->primeiro = (struct no *)malloc(sizeof (struct no));
	L->ultimo = L->primeiro;
	L->primeiro->prox = NULL;
}

void insere_lista(struct lsp x, struct lista *L )
{
	L->ultimo->prox = (struct no*)malloc(sizeof(struct no));
	L->ultimo=L->ultimo->prox;
	L->ultimo->item = x;
	L->ultimo->prox = NULL;
}

struct no *Retira_Lista_Final(struct lista *L) //Retira o Elemento (Nó) do Final da Lista
{
	struct no *aux, *retorno;
	aux = L->primeiro;
	if (aux->prox == NULL)
	{
		printf("Lista Vazia");
		return(NULL);
	}
	else
	{
		while(aux->prox->prox != L->ultimo->prox )
			aux = aux->prox;
		retorno=L->ultimo;
		L->ultimo = aux;
		retorno=L->ultimo->prox;
		L->ultimo->prox = NULL;
		return(retorno);
	}
}
void insere_lista_ordem_decrescente(struct lsp x,struct lista *L )
{
	struct no *aux, *aux_busca;
	aux  = (struct no*)malloc(sizeof(struct no));
	aux->item = x;
```

```
	if(aux->item.Carga <= L->ultimo->item.Carga || L->primeiro->prox == NULL)
	{
		L->ultimo->prox = aux;
		L->ultimo=L->ultimo->prox;
		L->ultimo->prox = NULL;
	}
	else
	{
		aux_busca = L->primeiro;
		while (aux_busca->prox->item.Carga > aux->item.Carga)
			aux_busca = aux_busca->prox;
		aux->prox = aux_busca->prox;
		aux_busca->prox = aux;
	}
}
struct no *RetiraLSPdaLista(int L_Number, struct lista *L)
{
	struct no *aux, *retorno;
	aux = L->primeiro;
	while(aux->prox != NULL)
	{
		if(aux->prox->item.NumParOD == L_Number)
		{

			break;

		}
		else
		{	aux = aux->prox;}
	}
	retorno=aux->prox;
	if(retorno!=NULL && retorno->prox != NULL)
	{
		aux->prox = retorno->prox;
		//free(retorno);
	}
	else
	{
		L->ultimo = aux;
		L->ultimo->prox = NULL;
		//aux->prox = NULL;
	}
	return(retorno);
}

void DesestabeleceLSP(int L_Number, struct no *LSP)
{
	int nEnl, j;
	for (nEnl=1;nEnl<=LINKS;nEnl++);
	{
		for (j=1; j<= MaxSaltos; j++)
		{
			if (lnk[nEnl].DescEnlace == LSP->item.CaminhoEscolhido[j])
			{
```

```
                                printf("LSP a ser retirada: %d\n",LSP->item.NumParOD);
                                RetiraLSPdaLista(LSP->item.NumParOD, &lnk[nEnl].LLSPs);
                        }
                }
        }
}
void RetiraLSPsPreempted(struct no *LSP)
{
        struct no *retorno, *LSP_Preemptado;
        struct evchain *retono_cancel;
        int j, nEnl, w;
        fprintf(fp,"%da PREEMPCAO NECESSARIO\n", contador);
        contador++;
        while (LPreempted.primeiro->prox!= NULL)
        {
                retorno = Retira_Lista_Final(&LPreempted);
                cancelp_tkn(((*retorno).item).NumParOD);
                for(w=0;w<MaxClassType;w++)
                {
                        if(w==retorno->item.CT)
                        {
                                if (retorno->item.preeemptadaPorDebito==1)
                                        N_LSPs_Preempted_Debt[w]++;
                                else
                                        N_LSPs_Preempted[w]++;
                        }
                }
                if(retorno->item.preeemptadaPorDebito==1)
                        Total_LSPs_Preempted_Debt++;
                else
                        Total_LSPs_Preempted++;
                fprintf(fp,"LSP%d(%d-%d)(CT=%d):      %.2lf      -      Preemptptada\n",retorno-
>item.NumParOD,retorno->item.src, retorno->item.dest,retorno->item.CT,retorno->item.Carga);
                for (nEnl=1;nEnl<=LINKS;nEnl++)
                {
                        for (j=1; j<= MaxSaltos; j++)
                        {
                                if  (lnk[nEnl].DescEnlace  ==  retorno->item.CaminhoEscolhido[j])

                                        LSP_Preemptado          =          RetiraLSPdaLista(retorno-
>item.NumParOD,&lnk[nEnl].LLSPs); // ESSA FUNÇÃO TEM QUE SER IMPLEMENTADA
                        }

                }
        }
        fprintf(fp,"\n\n");
}
void RetiraLSPDesestablish(int LSPNumber,struct no *LSP)
{
        struct no *retorno, *LSP_Desestabelecido;
        struct evchain *retono_cancel;
        int j, i, nEnl;
        for (nEnl=1;nEnl<=LINKS;nEnl++)
        {
                for (j=1; j<= MaxSaltos; j++)
```

```
		{
			if (lnk[nEnl].DescEnlace == LSP->item.CaminhoEscolhido[j])
			{
				LSP_Desestabelecido = RetiraLSPdaLista(LSPNumber,&lnk[nEnl].LLSPs);
				if(LSP_Desestabelecido!=NULL)
				{
					printf("LSP%d foi desestabelecida\n",LSP_Desestabelecido->item.NumParOD);
					fprintf(fp,"LSP%d(%d-%d)(CT=%d): %.2lf - desestabelecida\n",LSP_Desestabelecido->item.NumParOD,LSP_Desestabelecido->item.src, LSP_Desestabelecido->item.dest,LSP_Desestabelecido->item.CT,LSP_Desestabelecido->item.Carga);
				}
			}
		}
	}
}
void insere_LSP_TodasListas (struct no *retorno, int caminhoatual)
{
	int j, nEnl;
	for (j=1; j<= MaxSaltos; j++)
	{
		for (nEnl=1;nEnl<=LINKS;nEnl++)
		{
			if (lnk[nEnl].DescEnlace == retorno->item.MatCaminhos[caminhoatual][j])
				insere_lista (retorno->item, &lnk[nEnl].LPreempted);
		}
	}
}
void Atualiza_BCAUXs_porEnlace_posPreempcao(struct no *LSP, int n, double BandadasLSPsPreemptadas)
{
	int k, j, nEnl;
	for (k=1; k<= MaxSaltos; k++)
	{
		for (nEnl=1;nEnl<=LINKS;nEnl++)
		{
			if (lnk[nEnl].DescEnlace == LSP->item.CaminhoEscolhido[k])
			{
				for(j=0; j<=n; j++)
					lnk[nEnl].BCAcumuladoAUX[j] = lnk[nEnl].BCAcumuladoAUX[j] - BandadasLSPsPreemptadas;
			}
		}
	}
}
void Atualiza_BCs_porEnlace_posDesestabelecimento(struct no *LSP)
{
	int w, j, nEnl;
	for (nEnl=1;nEnl<=LINKS;nEnl++) // Atualizando a variável auxiliar "BCAcumulado[n]" o estabelecimento
	{
		for(j=0; j<MaxSaltos; j++)
		{
```

```
                        if (lnk[nEnl].DescEnlace == LSP->item.CaminhoEscolhido[j])
                        {
                                for(w=0; w<=LSP->item.CT; w++)
                                {
                                        if(lnk[nEnl].BCAcumuladoAUX[w] - LSP->item.Carga>0)
                                        {
                                                //printf("Link%d - BC pos Desestab%d = %.0lf\n",
nEnl, w, lnk[nEnl].BCAcumulado[w]);
                                                lnk[nEnl].BCAcumuladoAUX[w]                     =
lnk[nEnl].BCAcumulado[w] - LSP->item.Carga;
                                                //lnk[nEnl].BCAcumuladoAUX[w]                   =
lnk[nEnl].BCAcumuladoAUX[w] - LSP->item.Carga;
                                        }
                                        else
                                                lnk[nEnl].BCAcumuladoAUX[w] = 0;
                                        lnk[nEnl].BCAcumulado[w]                        =
lnk[nEnl].BCAcumuladoAUX[w];
                                }
                        }
                }
        }
}
void Atualiza_BCs_porEnlace_posNovoEstabelecimento() // Calcula a banda acumulada atual por BC
no enlace "nEnl"
{
        int j, nEnl;
        for (nEnl=1;nEnl<=LINKS;nEnl++) // Atualizando a variável auxiliar "BCAcumulado[n]" o
estabelecimento
        {
                for(j=0; j<MaxClassType; j++)
                        lnk[nEnl].BCAcumulado[j] = lnk[nEnl].BCAcumuladoAUX[j];
        }
}
void Atualiza_CargadosEnlaces()
{
        int nEnl;
        for(nEnl=1;nEnl<=LINKS;nEnl++)
        {
                lnk[nEnl].CargaEnlaceAtual =  lnk[nEnl].BCAcumulado[0];
                lnk[nEnl].CargaResidual = lnk[nEnl].CargaEnlace - lnk[nEnl].CargaEnlaceAtual;
        }
}
void Retira_Lista_Elemento_TodasListas(struct no *LSP)
{
        struct no *aux, *retorno;
        int nEnl;
        int k;
        for     (k=1;k<=MaxH;k++)
        {
                for (nEnl=1;nEnl<=LINKS;nEnl++)
                {
                        aux = lnk[nEnl].CT[LSP->item.CT].LPreemptebla[k].primeiro;
                        while(aux->prox != NULL)
                        {
                                if(aux->prox->item.NumParOD == LSP->item.NumParOD)
```

```
                                        {
                                                lnk[nEnl].BandaPreemptadaCT[LSP->item.CT]                =
lnk[nEnl].BandaPreemptadaCT[LSP->item.CT] + LSP->item.Carga;
                                                break;
                                        }
                                        aux = aux->prox;
                                }
                                retorno=aux;
                                if (retorno!=NULL)
                                {
                                        if(retorno->prox != NULL)
                                        {
                                                retorno=aux->prox;
                                                aux->prox = retorno->prox;
                                        }
                                        else
                                                aux->prox = NULL;
                                }
                        }
                }
        }
void Cria_ListaPreemptaveis_porEnlaceporCT(struct no *LSP, int Indice_Melhor_Caminho)
{
        int i, p, nEnl;
        struct no *aux;
        for(p=1;p<=MaxSaltos;p++)
        {
                for (nEnl=1;nEnl<=LINKS;nEnl++)
                {
                        if(LSP->item.MatCaminhos[Indice_Melhor_Caminho][p]                ==
lnk[nEnl].DescEnlace)
                        {
                                for(i=0;i<=MaxH;i++)
                                {
                                        aux = lnk[nEnl].LLSPs.primeiro;
                                        while(aux->prox!=NULL)
                                        {
                                                if (aux->prox->item.H == i)
                                                        //insere_lista_ordem_crescente    (aux->prox-
>item,&lnk[nEnl].CT[aux->prox->item.CT].LPreemptebla[i]);
                                                        insere_lista_ordem_decrescente    (aux->prox-
>item,&lnk[nEnl].CT[aux->prox->item.CT].LPreemptebla[i]);
                                                aux = aux->prox;
                                        }
                                }
                        }
                }
        }
void imprime_LSPs_Preemptados(struct lista *L) // Imprime elemento
{
        struct no *aux;
        aux = L->primeiro;
        while(aux->prox!=NULL)
        {
```

```
                printf("LSP%d:CT%d -> ", aux->prox->item.NumParOD, aux->prox->item.CT);
                aux = aux->prox;
        }
}
void imprime_BCAcumulado()
{
        int nEnl, i;
        fprintf(fp, "---------- Valor Atual dos BCs por Enlace ----------\n\n");
        for(nEnl=1;nEnl<=LINKS;nEnl++)
        {
                fprintf(fp,"------- Enlace%d -------\n", lnk[nEnl].NumEnlace);
                for(i=0;i<MaxClassType;i++)
                {
                        //fprintf(fp,"BandaCT[%d] = %lf\n", i, lnk[nEnl].BCPos[i]);
                        if(i!=MaxClassType-1)
                                fprintf(fp,"BC[%d] = %.0lf(%.2lf por cento) | CT[%d] = %lf(%.2lf
por cento)\n", i, lnk[nEnl].BCAcumulado[i], 100*lnk[nEnl].BCAcumulado[i]/lnk[nEnl].CargaEnlace,
i,      lnk[nEnl].BCAcumulado[i]-lnk[nEnl].BCAcumulado[i+1],      100*(lnk[nEnl].BCAcumulado[i]-
lnk[nEnl].BCAcumulado[i+1])/lnk[nEnl].CargaEnlace);
                        else
                                fprintf(fp,"BC[%d] = %.0lf(%.2lf por cento) | CT[%d] = %lf(%.2lf
por cento)\n", i, lnk[nEnl].BCAcumulado[i],100*lnk[nEnl].BCAcumulado[i]/lnk[nEnl].CargaEnlace,
i, lnk[nEnl].BCAcumulado[i], 100*lnk[nEnl].BCAcumulado[i]/lnk[nEnl].CargaEnlace);
                }
                fprintf(fp,"\n");
        }
        fprintf(fp, "\n----------------------------------------------------------------\n");
}
void imprime_lista(struct lista *L) // Imprime elemento
{
        struct no *aux;
        aux = L->primeiro->prox;
        if (aux == NULL)
        {
        fprintf(fp, "Lista Vazia");
        }
        else
        {
                while(aux!=NULL)
                {
                //printf("%d -> ", aux->item.dest);
                fprintf(fp, "LSP%d: %.3lf(%d-%d)(CT=%d) -> ", aux->item.NumParOD, aux-
>item.Carga, aux->item.src, aux->item.dest, aux->item.CT);
                aux = aux->prox;
                }
        }
}
int Verify_RDM_Preemption_Need (struct no *LSP, int Indice_Melhor_Caminho)
{
        struct no *aux, *retorno;
        int Prept = 0;
        int i,n, j, nEnl, w;
        int auxbraek=0;
        double BandaAcimaCT;
```

```
	Cria_ListaPreemptaveis_porEnlaceporCT(LSP, Indice_Melhor_Caminho);
	for (nEnl=1;nEnl<=LINKS;nEnl++)
	{
		for (j=1; j<= MaxSaltos; j++)
		{
			if (lnk[nEnl].DescEnlace == LSP->item.MatCaminhos[Indice_Melhor_Caminho][j])
			{
				insere_lista(LSP->item,&lnk[nEnl].LLSPs);
				for(n=0;n<MaxClassType;n++)
				{
					if(n<=LSP->item.CT)
						lnk[nEnl].BCAcumuladoAUX[n] = lnk[nEnl].BCAcumulado[n] + LSP->item.Carga;
					else
						lnk[nEnl].BCAcumuladoAUX[n] = lnk[nEnl].BCAcumulado[n];
				}
			}
		}
	}
	for (nEnl=1;nEnl<=LINKS;nEnl++)
	{
		fprintf(fp,"Enlace%d \n", nEnl);
		imprime_lista(&lnk[nEnl].LLSPs);
		fprintf(fp, "\n");
	}
	for (nEnl=1;nEnl<=LINKS;nEnl++)
	{
		for (j=1; j<= MaxSaltos; j++)
		{
			if (lnk[nEnl].DescEnlace == LSP->item.MatCaminhos[Indice_Melhor_Caminho][j])
			{
				for(n=LSP->item.CT-1;n>=0;n--)
				{
					BandaAcimaCT = lnk[nEnl].BCAcumuladoAUX[n] - ((BC[n]/BC[0])*lnk[nEnl].CargaEnlace);
					if(BandaAcimaCT>0)
					{
						Prept = 1;
						for(i=0;i<=MaxH;i++)
						{
							aux = lnk[nEnl].CT[n].LPreemptebla[i].primeiro;
							while(aux->prox!=NULL)
							{
								if(BandaAcimaCT <= lnk[nEnl].BandaPreemptada)
								{
									break;
									auxbraek=1;
								}
```

```
                                                retorno =
Retira_Lista_Final(&lnk[nEnl].CT[n].LPreemptebla[i]);
                                                Retira_Lista_Elemento_TodasListas(r
                                                etorno);
                                                insere_lista (retorno->item,
&LPreempted);
                                                lnk[nEnl].TotalBandaPreemptada =
lnk[nEnl].TotalBandaPreemptada + retorno->item.Carga;
                                                lnk[nEnl].BandaPreemptada =
lnk[nEnl].BandaPreemptada + retorno->item.Carga;
                                                Atualiza_BCAUXs_porEnlace_posPr
                                        eempcao(retorno, n, retorno->item.Carga);
                                                if(auxbraek==1)
                                                        break;
                                        }
                                }
                        }
                        lnk[nEnl].BandaPreemptada=0; // Reinicializa a variável
BandaPreemptada para futuros processamentos
                }
        }
    }
    return(Prept);
}
int     Verify_RDM_With_Debt_and_Reduct_Preemption_Need     (struct     no     *LSP,     int
Indice_Melhor_Caminho, int limiar)// Heurística de Adaptação que leva em conta o número de
Enlaces da Rede (Centralizada ou não? Já que o tratamento continua enlace por enlace)
{

    struct no *aux, *retorno;
    int Prept = 0;
    int i,n, j, nEnl, w;
    int auxbraek=0;
    double BandaAcimaCT;
    Cria_ListaPreemptaveis_porEnlaceporCT(LSP, Indice_Melhor_Caminho);
    for (nEnl=1;nEnl<=LINKS;nEnl++)
    {
        for (j=1; j<= MaxSaltos; j++)
        {
            if          (lnk[nEnl].DescEnlace          ==          LSP-
>item.MatCaminhos[Indice_Melhor_Caminho][j])
            {
                insere_lista(LSP->item,&lnk[nEnl].LLSPs);
                for(n=0;n<MaxClassType;n++)
                {
                    if(n<=LSP->item.CT)
                        lnk[nEnl].BCAcumuladoAUX[n]          =
lnk[nEnl].BCAcumulado[n] + LSP->item.Carga;
                    else
                        lnk[nEnl].BCAcumuladoAUX[n]          =
lnk[nEnl].BCAcumulado[n];
                }
            }
        }
```

```
	}
	for (nEnl=1;nEnl<=LINKS;nEnl++)
	{
		fprintf(fp,"Enlace%d \n", nEnl);
		imprime_lista(&lnk[nEnl].LLSPs);
		fprintf(fp, "\n");
	}
	for (nEnl=1;nEnl<=LINKS;nEnl++)
	{
		for (j=1; j<= MaxSaltos; j++)
		{
			if (lnk[nEnl].DescEnlace == LSP->item.MatCaminhos[Indice_Melhor_Caminho][j])
			{
				if (lnk[nEnl].BCAcumuladoAUX[0]>lnk[nEnl].CargaEnlace)
				{
					double emprestimos[MaxClassType];
					int r;
					for(r=MaxClassType-1;r>=0;r--)
					{
						if (r==MaxClassType-1)

	emprestimos[r]=((BC[r]/BC[0])*lnk[nEnl].CargaEnlace)-(lnk[nEnl].BCAcumulado[r])<0?((BC[r]/BC[0])*lnk[nEnl].CargaEnlace)-(lnk[nEnl].BCAcumulado[r]):0;
						else if (r==0)
							emprestimos[r]=0;
						else

	emprestimos[r]=((BC[r]/BC[0])*lnk[nEnl].CargaEnlace)-(lnk[nEnl].BCAcumulado[r]+emprestimos[r+1])<0?((BC[r]/BC[0])*lnk[nEnl].CargaEnlace)-(lnk[nEnl].BCAcumulado[r]+emprestimos[r+1]):0;
					}
					for(n=0;MaxClassType>n;n++)
					{

						BandaAcimaCT = -emprestimos[n];
						if(BandaAcimaCT >(lnk[nEnl].BCAcumuladoAUX[0] - lnk[nEnl].CargaEnlace))

	BandaAcimaCT=(lnk[nEnl].BCAcumuladoAUX[0] - lnk[nEnl].CargaEnlace);
						if(BandaAcimaCT>0)
						{
							Prept = 1;
								for(i=0;i<=MaxH;i++)
								{
									aux = lnk[nEnl].CT[n].LPreemptebla[i].primeiro;
									while(aux->prox!=NULL)
									{
										if(BandaAcimaCT <= lnk[nEnl].BandaPreemptada)
									{
										break;
										auxbraek=1;
```

```
                                                                }

        if((lnk[nEnl].CT[n].LPreemptebla[i].ultimo->item.Carga-(BandaAcimaCT -
lnk[nEnl].BandaPreemptada)) >= (lnk[nEnl].CT[n].LPreemptebla[i].ultimo->item.Carga*limiar/100))
                                                                {

        lnk[nEnl].CT[n].LPreemptebla[i].ultimo-
>item.Carga=(lnk[nEnl].CT[n].LPreemptebla[i].ultimo->item.Carga-(BandaAcimaCT -
lnk[nEnl].BandaPreemptada));

        Atualiza_BCAUXs_porEnlace_posPreempcao(lnk[nEnl].CT[n].LPreemptebla[i].ultimo,      n,
BandaAcimaCT - lnk[nEnl].BandaPreemptada);

        lnk[nEnl].TotalBandaPreemptada += BandaAcimaCT - lnk[nEnl].BandaPreemptada;

        lnk[nEnl].BandaPreemptada += BandaAcimaCT - lnk[nEnl].BandaPreemptada;
                                                                }
                                                                else
                                                                {
                                                                        retorno                 =
Retira_Lista_Final(&lnk[nEnl].CT[n].LPreemptebla[i]);

        Retira_Lista_Elemento_TodasListas(retorno);
                                                                        retorno-
>item.preeemptadaPorDebito=1;
                                                                        insere_lista    (retorno-
>item, &LPreempted);

        lnk[nEnl].TotalBandaPreemptada = lnk[nEnl].TotalBandaPreemptada + retorno->item.Carga;

        lnk[nEnl].BandaPreemptada = lnk[nEnl].BandaPreemptada + retorno->item.Carga;

        Atualiza_BCAUXs_porEnlace_posPreempcao(retorno, n, retorno->item.Carga);
                                                                }
                                                                if(auxbraek==1)
                                                                        break;

                                                                }
                                                        }
                                                }
                                                lnk[nEnl].BandaPreemptada=0;
                                        }
                                }
                                if (lnk[nEnl].BCAcumuladoAUX[0]>lnk[nEnl].CargaEnlace)
                                {
                                        for(n=LSP->item.CT-1;n>=0;n--)
                                        {
                                                BandaAcimaCT = lnk[nEnl].BCAcumuladoAUX[n] -
((BC[n]/BC[0])*lnk[nEnl].CargaEnlace);
                                                if(BandaAcimaCT
>(lnk[nEnl].BCAcumuladoAUX[0] - lnk[nEnl].CargaEnlace))

        BandaAcimaCT=(lnk[nEnl].BCAcumuladoAUX[0] - lnk[nEnl].CargaEnlace);
                                                if(BandaAcimaCT>0)
                                                {
```

```
                                                Prept = 1;
                                                        for(i=0;i<=MaxH;i++)
                                                        {
                                                                aux                             =
lnk[nEnl].CT[n].LPreemptebla[i].primeiro;
                                                                while(aux->prox!=NULL)
                                                                {
                                                                        if(BandaAcimaCT <=
lnk[nEnl].BandaPreemptada)
                                                                {
                                                                        break;
                                                                        auxbraek=1;
                                                                }

        if((lnk[nEnl].CT[n].LPreemptebla[i].ultimo->item.Carga-(BandaAcimaCT                    -
lnk[nEnl].BandaPreemptada)) >= (lnk[nEnl].CT[n].LPreemptebla[i].ultimo->item.Carga*limiar/100))
                                                                {

        lnk[nEnl].CT[n].LPreemptebla[i].ultimo-
>item.Carga=(lnk[nEnl].CT[n].LPreemptebla[i].ultimo->item.Carga-(BandaAcimaCT                    -
lnk[nEnl].BandaPreemptada));

        Atualiza_BCAUXs_porEnlace_posPreempcao(lnk[nEnl].CT[n].LPreemptebla[i].ultimo,      n,
BandaAcimaCT - lnk[nEnl].BandaPreemptada);

        lnk[nEnl].TotalBandaPreemptada += BandaAcimaCT - lnk[nEnl].BandaPreemptada;

        lnk[nEnl].BandaPreemptada += BandaAcimaCT - lnk[nEnl].BandaPreemptada;
                                                                }
                                                                else
                                                                {
                                                                        retorno                 =
Retira_Lista_Final(&lnk[nEnl].CT[n].LPreemptebla[i]);

        Retira_Lista_Elemento_TodasListas(retorno);
                                                                        insere_lista    (retorno-
>item, &LPreempted);

        lnk[nEnl].TotalBandaPreemptada = lnk[nEnl].TotalBandaPreemptada + retorno->item.Carga;

        lnk[nEnl].BandaPreemptada = lnk[nEnl].BandaPreemptada + retorno->item.Carga;

        Atualiza_BCAUXs_porEnlace_posPreempcao(retorno, n, retorno->item.Carga);
                                                                }
                                                                if(auxbraek==1)
                                                                        break;

                                                                }
                                                        }
                                                }
                                                lnk[nEnl].BandaPreemptada=0;
                                        }
                                }
                        }
                }
```

```
	}

	return(Prept);
}


/********* ALGORITMOS PARA SELEÇÃO DE CAMINHOS **********/
int TryPath_CSPF_WithAllocCTSharing (struct no *LSP, int limiar)
{
	double menorBorrowedBand=0, menorValorBCDisponivel=0;
	double maior = 0;
	int establish=0;
	int i,j,w,k, Prept;
	int nEnl;
	int Indice_Melhor_Caminho=0;
	int num_hops_caminho =0;
	int EstabelecimentoPossivel=0;
	double Vetor_Menor_BorrowedBand_por_Caminho[MaxCaminhos+1];
	double TotalFreeBand=0;
	double TotalFreeBandWithDebt=0;
	double menorBorrowedBandWithDebt=0;
	double Vetor_Menor_BorrowedBand_With_Debt_por_Caminho[MaxCaminhos+1];
	for (nEnl=1;nEnl<=LINKS;nEnl++)
	{
		for(k=0;k<MaxClassType;k++)
		{
			for(i=0;i<=MaxH;i++)
				inicia_lista(&lnk[nEnl].CT[k].LPreemptebla[i]);
		}
	}
	for(i=1; i<= MaxCaminhos; i++)
	{
		menorBorrowedBandWithDebt=DBL_MAX;
		menorBorrowedBand=DBL_MAX;
		for (j=1; j<= MaxSaltos; j++)
		{
			for (nEnl=1;nEnl<=LINKS;nEnl++)
			{
				if (lnk[nEnl].DescEnlace == LSP->item.MatCaminhos[i][j])
				{
					double emprestimo=0;
					double emprestimoCTSuperiores=0;
					double emprestimos[MaxClassType];
					double emprestimos_real[MaxClassType];
					int r;
					for(r=MaxClassType-1;r>=0;r--)
					{
						if (r==MaxClassType-1)

	emprestimos[r]=((BC[r]/BC[0])*lnk[nEnl].CargaEnlace)-
(lnk[nEnl].BCAcumulado[r])<0?(lnk[nEnl].BCAcumulado[r])-
((BC[r]/BC[0])*lnk[nEnl].CargaEnlace):0;
						else if (r==0)
							emprestimos[r]=0;
						else
```

```
        emprestimos[r]=((BC[r]/BC[0])*lnk[nEnl].CargaEnlace)-(lnk[nEnl].BCAcumulado[r]-
emprestimos[r+1])<0?(lnk[nEnl].BCAcumulado[r]-emprestimos[r+1])-
((BC[r]/BC[0])*lnk[nEnl].CargaEnlace):0;
                                        }
                                        for(r=MaxClassType-1;r>LSP->item.CT;r--)
                                                emprestimoCTSuperiores+=emprestimos[r];

                                        if(lnk[nEnl].BCAcumulado[LSP->item.CT]-
emprestimoCTSuperiores > (BC[LSP->item.CT]/BC[0])*lnk[nEnl].CargaEnlace)
                                        {
                                                for(r=LSP->item.CT-1;r>=0;r--)
                                                        emprestimo+=emprestimos[r];
                                                TotalFreeBandWithDebt = lnk[nEnl].CargaEnlace -
lnk[nEnl].CargaEnlaceAtual + emprestimo;
                                        }
                                        else
                                        {
                                                double naoUtilizadoBC=0;
                                                for(r=MaxClassType-1;r>=0;r--)
                                                        emprestimo+=emprestimos[r];
                                                TotalFreeBandWithDebt = lnk[nEnl].CargaEnlace -
lnk[nEnl].CargaEnlaceAtual + emprestimo;
                                                naoUtilizadoBC = ((BC[LSP-
>item.CT]/BC[0])*lnk[nEnl].CargaEnlace) - lnk[nEnl].BCAcumulado[LSP->item.CT];
                                                if(naoUtilizadoBC>TotalFreeBandWithDebt)
                                                        TotalFreeBandWithDebt=naoUtilizadoBC;
                                        }
                                        TotalFreeBand = lnk[nEnl].CargaEnlace -
lnk[nEnl].CargaEnlaceAtual;
                                        if(menorBorrowedBand > TotalFreeBand)
                                                menorBorrowedBand = TotalFreeBand;
                                        if(menorBorrowedBandWithDebt > TotalFreeBandWithDebt)
                                                menorBorrowedBandWithDebt =
TotalFreeBandWithDebt;
                                }
                                TotalFreeBand=0;
                        }

                }
                Vetor_Menor_BorrowedBand_por_Caminho[i] = menorBorrowedBand;
                Vetor_Menor_BorrowedBand_With_Debt_por_Caminho[i] =
menorBorrowedBandWithDebt;
                menorBorrowedBand  = menorValorBCDisponivel = 0;
        }
        for(i=1;i<=LSP->item.NumCaminhosOD; i++)
        {
                if(Vetor_Menor_BorrowedBand_por_Caminho[i]!=DBL_MAX && LSP->item.Carga
<= Vetor_Menor_BorrowedBand_por_Caminho[i])
                {
                        EstabelecimentoPossivel =1;
                        if(EstabelecimentoPossivel==1)
                        {
                                Indice_Melhor_Caminho = i;
                                break;
```

```
                        }
                }
        }
        if(EstabelecimentoPossivel==0)
        {
                for(i=1;i<=LSP->item.NumCaminhosOD; i++)
                {
                        if(Vetor_Menor_BorrowedBand_With_Debt_por_Caminho[i]!=DBL_MAX
&& LSP->item.Carga <= Vetor_Menor_BorrowedBand_With_Debt_por_Caminho[i])
                        {
                                EstabelecimentoPossivel =1;
                                if(EstabelecimentoPossivel==1)
                                {
                                        Indice_Melhor_Caminho = i;
                                        break;
                                }
                        }
                }

        }
        if(EstabelecimentoPossivel==0)
        {
                insere_lista(LSP->item, &LBlocked);
                for(w=0;w<MaxClassType;w++)
                {
                        if(w==LSP->item.CT)
                                N_LSPs_Blocked[w]++;
                }
                Total_LSPs_Blocked++;
                return(establish);
        }
        else
        {
                fprintf(fp,"LSP%d(%d-%d)(CT=%d):  %.2lf  -  Estabelecida\n",LSP-
>item.NumParOD,LSP->item.src, LSP->item.dest,LSP->item.CT,LSP->item.Carga);
                fprintf(fp, "\n");
                establish=1;
                for (w=1;w<=MaxSaltos;w++)
                {
                        LSP->item.CaminhoEscolhido[w]  =  LSP-
>item.MatCaminhos[Indice_Melhor_Caminho][w];
                        LSP->item.NumHops++;
                }
                Prept  =  Verify_RDM_With_Debt_and_Reduct_Preemption_Need  (LSP,
Indice_Melhor_Caminho, limiar);
                if(Prept == 1)
                        RetiraLSPsPreempted(LSP);
                Atualiza_BCs_porEnlace_posNovoEstabelecimento();
                Atualiza_CargadosEnlaces();
                imprime_BCAcumulado();
                return(establish);
        }
}

int TryPath_CSPF_WithPreemption (struct no *LSP)
```

```
{
	int auxbreak1=0, auxbreak2=0, establish=0;
	int i,j,w,k, Prept;
	int nEnl;
	int Indice_Melhor_Caminho=-1;
	int num_hops_caminho =0;
	for (nEnl=1;nEnl<=LINKS;nEnl++)
	{
		for(k=0;k<MaxClassType;k++)
		{
			for(i=0;i<=MaxH;i++)
				inicia_lista(&lnk[nEnl].CT[k].LPreemptebla[i]);
		}
	}
	for(i=1;i<=LSP->item.NumCaminhosOD; i++)
	{
		for (j=1; j<= /*1*/MaxSaltos; j++)
		{
			for (nEnl=1;nEnl<=LINKS;nEnl++)
			{
				if (lnk[nEnl].DescEnlace == LSP->item.MatCaminhos[i][j])
				{
					if(LSP->item.Carga + lnk[nEnl].BCAcumulado[LSP-
>item.CT] > (BC[LSP->item.CT]/BC[0])*lnk[nEnl].CargaEnlace)
					{
						auxbreak1=1;
						break;
					}
					else
						auxbreak1=0;
				}
			}
			if(auxbreak1==1)
			{
						break;
			}
		}
		if(auxbreak1==0)
		{
			fprintf(fp,"LSP%d(%d-%d)(CT=%d): %.2lf - Estabelecida\n",LSP-
>item.NumParOD,LSP->item.src, LSP->item.dest,LSP->item.CT,LSP->item.Carga);
			fprintf(fp, "\n");
			Indice_Melhor_Caminho = i;
			establish=1;
			for (w=1;w<=MaxSaltos;w++)
			{
				LSP->item.CaminhoEscolhido[w] = LSP-
>item.MatCaminhos[Indice_Melhor_Caminho][w];
				LSP->item.NumHops++;
			}
			Prept = Verify_RDM_Preemption_Need (LSP, Indice_Melhor_Caminho);
			break;
		}
	}
	if(establish==1)
```

```
	{
		if(Prept == 1)
		{
			RetiraLSPsPreempted(LSP);
		}
		Atualiza_BCs_porEnlace_posNovoEstabelecimento();
		Atualiza_CargadosEnlaces();
		imprime_BCAcumulado();
	}
	else
	{
		for(w=0;w<MaxClassType;w++)
		{
			if(w==LSP->item.CT)
				N_LSPs_Blocked[w]++;
		}
		Total_LSPs_Blocked++;
	}
	return(establish);
}

/****************** GERAÇÃO DE NÚMEROS ALEATORIOS ********************/
real ranf()
{
	real num_alea;
	num_alea = rand ();
	if ( num_alea < 0.)
	{
		printf ("\n\nFc ranf() - Erro grave no gerador do S.O.");
		exit(0);
	}
	while (num_alea == 0.)

	{
		num_alea = rand ();

	}
	return ( (float) num_alea / RAND_MAX );
 }

/*************** FUNÇÕES ADAPTADAS DO SMPL ******************/
struct evchain *cancelp_tkn(int tkn)
{
	struct evchain *evc, *evc_tkn_srv;
	if ( tkn == 0 )
	{
		printf ("\n Erro - fc cancelp_tkn - erro fatal token = 0 ");
		exit (0);
	}
	if ( (sim[sn].evc_begin == NULL) && (sim[sn].evc_end == NULL) )
	{
		printf ("\n Erro - fc cancelp - cadeia de eventos vazia");
		exit (0);
	}
	evc = sim[sn].evc_begin;
```

```
	evc_tkn_srv = NULL;
	while (evc != NULL)
	{
		if ( evc->ev_tkn == tkn)
		{
			evc_tkn_srv = evc;
			break;
		}
		evc = evc->ev_next;
	}
	if ( evc_tkn_srv == NULL )
	{
		printf ("\n Erro - fc cancelp - tkn em servico nao encontrada na cadeia de eventos");
		exit (0);
	}
	if ((evc_tkn_srv->ev_prior == NULL) && (evc_tkn_srv->ev_next == NULL))
	{
		sim[sn].evc_begin = NULL;
		sim[sn].evc_end = NULL;
		return (evc_tkn_srv);
	}
	if ((evc_tkn_srv->ev_prior == NULL) && (evc_tkn_srv->ev_next != NULL))
	{
		sim[sn].evc_begin = evc_tkn_srv->ev_next;
		evc_tkn_srv->ev_next->ev_prior = NULL;
		return (evc_tkn_srv);
	}
	if ((evc_tkn_srv->ev_prior != NULL) && (evc_tkn_srv->ev_next == NULL))
	{
		sim[sn].evc_end = evc_tkn_srv->ev_prior;
		evc_tkn_srv->ev_prior->ev_next = NULL;
		return (evc_tkn_srv);
	}
	if ((evc_tkn_srv->ev_prior != NULL) && (evc_tkn_srv->ev_next != NULL))
	{
		evc_tkn_srv->ev_prior->ev_next = evc_tkn_srv->ev_next;
		evc_tkn_srv->ev_next->ev_prior = evc_tkn_srv->ev_prior;
		return (evc_tkn_srv);
	}
	printf ("\n Erro - fc cancelp - elemento fora de opcoes de retirada");
	exit (0);
}

Init_evchain()
{
	sn = sn + 1;
	if ( sn > MAX_SIMULATIONS )
	{
		printf ("\n Erro - fc smpl - numero de simulacoes extrapolou
MAX_SIMULATIONS");
		exit (0);
	}
	sim[sn].evc_begin = NULL;
	sim[sn].evc_end = NULL;
	event = 0;
```

```
}

void schedulep(int ev, double te, int tkn, struct no *tkp)
{
        struct evchain *evc, *evc_aux;
        double st;

        st = clock + te;
        if ( tkn == 0 )
        {
                printf ("\n Erro - fc schedulep - erro fatal token = 0 ");
                exit (0);
        }
        if (te < 0.0)
        {
                printf ("\n Erro - fc schedule - tempo simulado menor que zero");
                exit (0);
        }
        evc = malloc(sizeof(struct evchain));
        if (evc == NULL)
        {
                printf ("\n Erro - fc schedulep - nao tem mais memoria para alocar para evento");
                exit (0);
        }
        evc->ev_time = st;
        evc->ev_tkn = tkn;
        evc->ev_tkn_p = tkp;
        evc->ev_type = ev;
        if ( (sim[sn].evc_begin == NULL) && (sim[sn].evc_end == NULL) )
        {
                sim[sn].evc_begin = evc;
                sim[sn].evc_end = evc;
                evc->ev_next = NULL;
                evc->ev_prior = NULL;
                return;
        }
        evc_aux = sim[sn].evc_end;
        if (( st > evc_aux->ev_time ) || ( st == evc_aux->ev_time ))
        {


                evc_aux->ev_next = evc;

                evc->ev_prior = evc_aux;

                evc->ev_next = NULL;
                sim[sn].evc_end = evc;
                return;
        }
        evc_aux = sim[sn].evc_begin;
        if ( st < evc_aux->ev_time )
        {
                evc_aux->ev_prior = evc;
                evc->ev_prior = NULL;
                evc->ev_next = evc_aux;
```

```
                sim[sn].evc_begin = evc;
                return;
        }
        if ( st == evc_aux->ev_time )
        {
                evc->ev_next = evc_aux->ev_next;
                evc->ev_prior = evc_aux;
                evc_aux->ev_next->ev_prior = evc;
                evc_aux->ev_next = evc;
                return;
        }
        evc_aux = sim[sn].evc_end;
        while (1)                       /* Insere na lista mas nao e o primeiro */
        {
                if (( st > evc_aux->ev_time ) || ( st == evc_aux->ev_time ))
                {
                        evc->ev_next = evc_aux->ev_next;
                        evc->ev_prior = evc_aux;
                        evc_aux->ev_next->ev_prior = evc;
                        evc_aux->ev_next = evc;
                        return;
                }
                evc_aux = evc_aux->ev_prior;
        }
        printf ("\n Erro - fc schedulep - erro no tempo nao inserido na lista de eventos");
        exit (0);
}

struct no *causep(int *ev, int *tkn)
{
        struct evchain *evc;
        struct no *tkp;


        evc = sim[sn].evc_begin;

        *tkn = evc->ev_tkn;
        *ev = evc->ev_type;
        clock = evc->ev_time;
        tkp = evc->ev_tkn_p;
        sim[sn].evc_begin = evc->ev_next;
        if (evc->ev_next == NULL)
        {
                sim[sn].evc_begin = NULL;
                sim[sn].evc_end = NULL;
        }
        else
        {
                evc->ev_next->ev_prior = NULL;
        }
        free(evc);
        return (tkp);

}
```

```
double simtime(void)
{
	return(clock);
}

void imprime_evchain() // Imprime elemento
{
	struct evchain *aux;
	aux = sim[1].evc_begin->ev_next;
	if (aux == NULL)
	{
		fprintf(fp, "Lista Vazia");
	}
	else
	{
		while(aux!=NULL)
		{
			fprintf(fp, "(Ev=%d|Time=%.2lf|LSP%d|CT%d|Banda=%.0lf) -> ", aux->ev_type, aux->ev_time, aux->ev_tkn_p->item.NumParOD, aux->ev_tkn_p->item.CT,aux->ev_tkn_p->item.Carga);
			aux = aux->ev_next;
		}
	}
}

/*************** DISTRIBUIICOES DE PROBABILIDADE UTILIZADAS *****************/

/*--------------  EXPONENTIAL RANDOM VARIATE GENERATOR  --------------*/
real expntl(real x)
{ /* 'expntl' returns a psuedo-random variate from a negative     */
  /* exponential distribution with mean x.                        */
  return(-x*log(ranf()));
}


/*------------  UNIFORM [a, b] RANDOM VARIATE GENERATOR  -------------*/
real uniform(real a, real b)
{ /* 'uniform' returns a psuedo-random variate from a uniform     */
  /* distribution with lower bound a and upper bound b.           */
  //if (a>b) then error(0,"uniform Argument Error: a > b");
  return(a+(b-a)*ranf());
}
```