FEEC Faculdade de Engenharia Elétrica e de Computação

Universidade Estadual de Campinas
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE TELEMÁTICA

Especificação Formal em SDL e Simulação de Protocolos que combinam a Técnica de *Label-Swapping* com o Roteamento de Camada 3 para o Transporte do Ip sobre o ATM

por

Nadia Adel Nassif
Bacharel em Ciências da Computação - Universidade Estadual de Londrina

Banca Examinadora :

Walter da Cunha Borelli (**Orientador**) DT/FEEC/UNICAMP,
Maurício Ferreira Magalhães DCA/FEEC/UNICAMP,
Juan Manuel Adan Coello II/PUCCAMP,
Ivanil S. Bonati (**Suplente**) DT/FEEC/UNICAMP

Tese apresentada à Faculdade de Engenharia Elétrica e de Computaçao da Universidade Estadual de Campinas, como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

5 de Março de 1999

Este exemplar corresponde a redação final da tese defendida por Nadia Adel Nassif e aprovada pela Comissão Julgada em 05/03/99
Orientador

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Nassif, Nadia Adel
N188e Especificação formal em SDL e simulação de protocolos que combinam a técnica de *Label-Swapping* com o roteamento de camada 3 para o transporte do Ip sobre o ATM. / Nadia Adel Nassif.--Campinas, SP: [s.n.], 1999.

Orientador: Walter da Cunha Borelli
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. SDL (Linguagem de programação de computador). 2. Redes de computação – Protocolos. 3. Simulação (Computadores). I. Borelli, Walter da Cunha. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Aos meus pais Fatma e Adel apesar da distância e da saudade, pelo incentivo, paciência, compreensão e amor.

Ao Adriano pelo companheirismo e por estar sempre presente durante o desenvolvimento deste trabalho.

Agradecimentos

Aos meus pais pelo incentivo, pelos conselhos, pela paciência, pelo amor e pela dedicação.

A minha irmã Magda pelo incentivo.

Ao Prof. Maurício Ferreira Magalhães pela atenção e pela ajuda nas discussões sobre os protocolos, que foram de grande importância para o desenvolvimento deste trabalho.

Aos administradores do laboratório do departamento de telemática, Jesus e Trettel, pela ajuda e esclarecimento de dúvidas técnicas sobre o Unix, Latex, SDT, etc.

Ao meu orientador, Prof. Walter da Cunha Borelli, pela oportunidade do mestrado.

Aos colegas que conheci durante as disciplinas cursadas e no laboratório do DT.

À Luciana e à Alessandra pela companhia na república.

Aos amigos Giuliano e Fábio pelos e-mails sempre tentando me acalmar.

Em especial ao Adriano pela amizade e companhia na Unicamp em vários dias de estudo e trabalho.

Resumo

A tese trata da especificação formal em SDL (*Specification and Description Language*) e da simulação através do SDT [1] (*SDL Design Tool*), de um classificador de fluxo X/Y usado no IpSwitching e, do LDP (*Label Distribution Protocol*) do MPLS (*MultiProtocol Label Switching*), partindo-se das recomendações do IETF (*Internet Engineering Task Force*).

Tanto o IpSwitching como o MPLS são protocolos que utilizam a combinação da técnica de *label-swapping* com o roteamento de camada 3, para transportar pacotes de um protocolo de rede sem conexão sobre uma tecnologia de comutação, como o IP sobre o ATM.

Com o objetivo de se avaliar a performance do IpSwithing em relação à variação dos parâmetros do classificador de fluxo X/Y, que é parte integrante do controlador de um IpSwitch, foram realizadas simulações com diferentes *traces* de tráfego IP.

Em relação ao MPLS é proposto uma especificação única que combina as características dos procedimentos de distribuição de *label* com a máquina de estado do LDP. O objetivo é analisar através de simulações, o comportamento de um LSR (*Label Switching Router*) em diferentes configurações, diante do recebimento de mensagens LDP e do estabelecimento de LSPs (*Label Switched Paths*).

Palavras Chaves : SDL - *Specification and Description Language*, SDT - *SDL Design Tool*, Especificação Formal, Simulação, MSC - *Message Sequence Chart*, IP sobre ATM, IpSwitching, Classificador de Fluxo, MPLS - *MultiProtocol Label Switching*, LDP - *Label Distribution Protocol*, *label-swapping*.

Publicação durante o Período : N.A. Nassif, W.C.Borelli, "A Combinação da Técnica de Orientação a Objetos OOA com a Linguagem Formal de Especificação SDL para o desenvolvimento de um Sistema de Banco de Dados", Anais do Terceiro Congresso Argentino de Ciência da Computação, CACIC97, La Plata-Argentina, 1997.

[1] Pacote SDT, versão 3.3 (*SDT Base, MSC Editor and Simulator*) : adquirido pelo DT/FEEC/UNICAMP através de um Projeto Temático FAPESP (Proc. 91/3660-0).

Abstract

The thesis deals with formal specification using SDL (Specification and Description Language) and the simulation using SDT [2] (SDL Design Tool) of the flow classifier X/Y used in IpSwitching, and of the LDP (Label Distribution Protocol) from MPLS (MultiProtocol Label Switching), based on the IETF (*Internet Engineering Task Force*) recomendations.

IpSwitching and MPLS are protocols that use the combination of the *label-swapping* technique with the layer 3 routing, forwarding connectionless traffic over a switching technology like IP over ATM.

In order to analize the IpSwitching performance varying the flow classifier X/Y paremeters, which is part of the IpSwitch controller, it was used different IP traffic traces on the simulations.

Concerning the MPLS it is proposed a single specification that combines the LDP procedures and LDP state machine. The goal is to verify the LSR (Label Switching Router) behaviour in recieving LDP messages and establishing LSPs (Label Switched Path), using different LSR configurations.

Key Words: SDL - Specification and Description Language, SDT - SDL Design Tool, Formal Specification, Simulation, MSC - Message Sequence Chart, IP over ATM, IpSwitching, Flow Classifier, MPLS - MultiProtocol Label Switching, LDP - Label Distribution Protocol, label-swapping.

Publications : N.A. Nassif, W.C.Borelli, "A Combinação da Técnica de Orientação a Objetos OOA com a Linguagem Formal de Especificação SDL para o desenvolvimento de um Sistema de Banco de Dados", Anais do Terceiro Congresso Argentino de Ciência da Computação, CACIC97, La Plata-Argentina, 1997.

[2]SDT Package, version 3.3 (SDT Base, MSC Editor and Simulator) : brought by DT/FEEC/UNICAMP of FAPESP Tematic Project (Proc. 91/3660-0).

Lista de Siglas e Acrônimos

AAL - *ATM Adaptation Layer*

ARIS - *Aggregate Route Based Ip Switching*

ATM - *Asynchronous Transfer Mode*

CIP - *Classical Ip*

CSR - *Cell Switched Router*

FEC - *Forwarding Equivalence Class*

GSMP - *Generic Switch Management Protocol*

IETF - *Internet Engineering Task Force*

IFMP - *Ipsilon Flow Management Protocol*

IP - *Internet Protocol*

LANE - *Lan Emulation*

LDP - *Label Distribution Protocol*

LLC - *Logical Link Control*

SNAP - *SubNetwork Attachment Point*

LSP - *Label Switched Path*

LSR - *Label Switching Router*

MPLS - *MultiProtocol Label Switching*

MPOA - *MutiProtocol Over ATM*

MSC - *Message Sequence Chart*

NBMA - *Non Broadcast Multiple Acess*

NHRP - *Next Hop Resolution Protocol*

OSPF - *Open Shortest Path First*

PDU - *Protocol Data Unit*

SDL - *Specification and Description Language*

SDT - *SDL Design Tool*

SVC - *Switched Virtual Connection*

TCP - *Transport Control Protocol*

UDP - *User Datagram Protocol*

VCI - *Virtual Circuit Identifier*

VPI - *Virtual Path Identifier*

camada 2 - a camada de protocolo abaixo da camada 3 e que oferece serviços usados pela camada 3. O transporte de pacotes quando é feito pela comutação de *labels*, ocorre na camada 2 não importando se o *label* examinado, é um VPI/VCI ATM, um DLCI Frame Relay ou um *label* MPLS.

camada de enlace - o mesmo que camada 2.

camada 3 - camada de protocolo no qual rodam o IP e os protocolos de roteamento associados.

camada de rede - o mesmo que camada 3.

fluxo - uma instância de fluxo de dados entre aplicações ou entre um nó origem e um nó destino.

label - identificador de tamanho fixo e curto usado para identificar uma FEC ou um fluxo Ip.

VC-merge - *VC-merge* é o processo no qual um comutador recebe células em vários VCIs de entrada e os transmite em uma único VCI de saída, sem que células de diferentes PDUs AAL5 fiquem entrelaçadas.

Índice

Capítulo 1

Introdução

1.1 Motivação e Objetivos

Nos últimos anos a Internet vem apresentando um crescimento exponencial. Este crescimento se deu em grande parte, devido à popularização de uma rede que, a princípio, era utilizada basicamente por pesquisadores e pela comunidade acadêmica.

Com o advento da WWW (*World-Wide-Web*) milhares de novos usuários se conectaram à rede pelo acesso residencial. O grande aumento no número de usuários gerou um grande crescimento na demanda de tráfego IP.

Há previsões de um crescimento da demanda de tráfego IP contínuo e cada vez maior com as novas tecnologias, como o transporte de voz e o comércio eletrônico [10].

Além do surgimento de novas tecnologias, há um grande desenvolvimento de aplicações multimídia e de tempo real, que fazem com que, a tendência seja de que cada usuário aumente sua própria demanda de tráfego. Este intenso tráfego está tornando difícil o transporte dos pacotes pelas redes tradicionais roteadas. O aumento do número de usuários e consequente aumento de tráfego, traz a necessidade de um maior número de roteadores e requerimentos como maior largura de banda. Os roteadores além de serem equipamentos de alto custo, acabam tornando-se pontos de congestionamento na rede.

Para tentar solucionar este problema, o tráfego IP começou a ser transportado pela rede ATM que tem despertado bastante interesse, devido a sua alta capacidade, escalabilidade de largura de banda, habilidade em suporte de tráfego multiserviço e alta velocidade dos equipamentos comutadores ATM.

Entretanto, o IP e o ATM utilizam-se de técnicas completamente diferentes para o transporte de dados. O IP (*Internet Protocol*) é um protocolo de camada de rede (camada 3) não orientado a conexão. Isto significa que os pacotes são roteados em cada ponto da rede conforme forem sendo transmitidos. A entrega das unidades de informação é baseada nos recursos da rede em cada ponto.

Já o ATM (*Asynchronous Transfer Mode*), é uma tecnologia de comutação orientada a conexão, onde o caminho por onde os dados serão transmitidos é estabelecido entre a origem e o destino antes que se inicie qualquer tipo de troca de informação.

Devido a estas diferenças entre a natureza não orientada a conexão do IP, e a natureza

orientada a conexão do ATM, o grande desafio do transporte IP sobre ATM, é justamente a utilização das vantagens de velocidade e capacidade de comutação do hardware ATM, sem sacrificar a flexibilidade da natureza sem conexão do IP. Isto significa, prover conectividade de camada 3, com performance de camada 2 [12].

Há várias propostas para se realizar o roteamento IP sobre ATM. Dois órgãos de padronização, o ATM Forum e o IETF (*Internet Engineering Task Force*), vêm apresentando diferentes propostas que diferem entre si, pela integração do IP com o ATM. Algumas das propostas são o IP Clássico [22], o NHRP [18] (*Next Hop Resolution Protocol*) e o MARS [9] (*MultiCast Resolution Address Server*) do IETF e, o LANE [3][6] (*Lan Emulation*) e o MPOA [7] (*MultiProtocol Over ATM*) do ATM Forum.

Além disso, existe a proposta de uma nova geração de protocolos que se utilizam da combinação da técnica de *label-swapping* e do roteamento de camada 3. Algumas das soluções são propostas proprietárias como o *Tag Switching* [36] da Cisco, o IpSwitching [14] da Ipsilon, o *Cell Switch Router* (CSR) [19][34] da Toshiba e o *Aggregate Route Based Ip Switching* (ARIS) [13] da IBM.

Além das propostas proprietárias, o MPLS *Working Group* do IETF está trabalhando no protocolo MPLS (*MultiProtocol Label Switching*) com o objetivo de padronizar uma tecnologia que combine a técnica de *label-swapping* com o roteamento de camada 3.

Este trabalho apresenta a especificação formal e a simulação do classificador de fluxo, parte integrante do controlador do IpSwitch (protocolo IpSwitching), e dos Procedimentos de Distribuição de Label e da Máquina de Estado do LDP (*Label Distribution Protocol*), o protocolo de distribuição de *labels* do MPLS.

As especificações foram realizadas utilizando-se a linguagem de especificação formal SDL92 (*Specification and Description Language*). A linguagem SDL92 é uma linguagem orientada a objetos. Evoluções de especificações em SDL foram realizadas neste trabalho através do uso do conceito de reutilização de código.

As simulações foram realizadas utilizando-se a ferramenta SDT [1] (*SDL Design Tool*) que possibilitou a geração de diagramas MSC [17] (*Message Sequence Chart*).

O objetivo do trabalho, além de realizar um estudo generalizado das novas propostas, é também, analisar as funções comportamentais e/ou o desempenho dos protocolos estudados, através das simulações diante de diferentes configurações das especificações.

1.2 Organização da Dissertação

Este trabalho esta organizado em capítulos e apêndices.

O segundo capítulo apresenta uma descrição das propostas do IETF (*Internet Engineering Task Force*) e do ATM Forum para o transporte do IP sobre ATM. São apresentados em mais detalhes os fundamentos dos protocolos IpSwitching e MPLS (*MultiProtocol Label Switching*) que são especificados e simulados neste trabalho.

[1] Pacote SDT, versão 3.3 (*SDT Base, MSC Editor and Simulator*) : adquirido pelo DT, FEEC/UNICAMP através de um Projeto Temático FAPESP (Proc. 91/3660-0).

O terceiro capítulo apresenta a especificação formal do classificador de fluxo presente no controlador de um IpSwitch. As simulações realizadas utilizam-se de *traces* de tráfego IP como entrada e, que estão disponíveis por ftp em *ita.ee.lbl.gov/traces*. Através da utilização destes *traces* nas simulações, além da parte comportamental do classificador de fluxo observada através dos diagramas MSC, pôde-se obter resultados estatísticos sobre o desempenho do protocolo como por exemplo : número total de pacotes processados pelo simulador, número total de pacotes comutados, número de fluxos detectados, número de fluxos selecionados para a comutação e, porcentagem de comutação.

O quarto capítulo apresenta em uma única especificação formal os procedimentos de distribuição de *labels* e a Máquina de Estado do LDP. O objetivo da especificação é analisar as funções comportamentais de um LSR (*Label Switching Router*) diante da troca de mensagens LDP com os LDP *peers* adjacentes e, a máquina de estado do LDP para cada LSP (*Label Switched Path*) a ser estabelecido. Além disso, o diagrama da máquina de estado do LDP para LSRs ATM definido em [24], foi expandido em 4 diagramas para abranger todas as configurações possíveis de um LSR : *DownStream* Independente, *DownStream* Ordenado, *DownStream-On-Demand* Independente e, *DownStream-On-Demand* Ordenado.

O quinto capítulo apresenta as conclusões obtidas no trabalho, assim como as contribuições e sugestões para trabalhos futuros na linha de protocolos que utilizam a técnica de *label swapping* combinado com o roteamento de camada 3.

O apêndice A apresenta uma descrição das variáveis da especificação do capítulo 4, para uma melhor compreensão dos resultados obtidos nas simulações.

O apêndice B apresenta resultados de simulações do capítulo 4 com informações detalhadas sobre os valores das variáveis após a ocorrência de cada evento, assim como a descrição do diagrama MSC gerado.

Capítulo 2

IP sobre ATM

2.1 Introdução

Com o sucesso mundial da rede Internet e consequentemente o aumento do número de usuários, o tráfego IP tem crescido exponencialmente durante os últimos anos, causando um proporcional aumento no número de roteadores nas redes tradicionais. Este crescimento sobre uma estrutura baseada em processamento intensivo para a busca da melhor rota, tem comprometido o desempenho das redes.

O IP (*Internet Protocol*) é um protocolo de camada de rede (camada de nível 3) não orientado a conexão que usa o sistema de entrega de pacotes *best-effort*.

A entrega das unidades de informação (datagramas) é baseada nos recursos da rede em cada ponto (*hop-by-hop*), conforme os dados forem sendo transmitidos. Este sistema de entrega possibilita por exemplo, o cálculo de uma nova rota, no caso de congestionamento ou falha na rede. A rede se compromete a tentar entregar cada datagrama individualmente no seu destino, entretanto, não é responsável pelo seu conteúdo, integridade e sequenciação. A recuperação de erro e sequenciamento são funções do nó receptor. Os datagramas são roteados independentemente uns dos outros, mesmo que apresentem o mesmo endereço destino [31].

Já o ATM [20] (*Asynchronous Transfer Mode*) é uma tecnologia de comutação orientada a conexão na qual, o caminho por onde os dados irão trafegar, é estabelecido antes que se comece qualquer transmissão de dados de uma origem a um destino. A rede ATM tem despertado bastante interesse devido à sua alta capacidade, escalabilidade de largura de banda e habilidade no suporte de tráfego multiserviço, além da alta velocidade dos equipamentos comutadores ATM.

Como o ATM é orientado a conexão, o grande desafio no transporte do IP sobre ATM é utilizar das vantagens de velocidade e capacidade de comutação do hardware ATM, sem sacrificar a escalabilidade e flexibilidade que vem da natureza sem conexão do IP. Isto significa prover conectividade de camada 3 com performance de camada 2.

Dois órgãos, o ATM Forum e o IETF (*Internet Engineering Task Force*), vêm trabalhando em propostas para a utilização do IP sobre o ATM. Estas propostas variam no nível de integração entre o IP e o ATM.

As primeiras propostas dos dois órgãos, o LANE [3][6] (*Lan Emulation*) e o MPOA [7] (*Multiprotocol Over ATM*) do ATM Forum, e o IP Clássico [22], o NHRP [18] (*Next Hop Resolution Protocol*) e o MARS [9] (*Multicast Address Resolution Server*) do IETF, utilizam o ATM como uma caixa preta que provê serviços bem definidos às camadas superiores. Esta abordagem tenta esconder a topologia da rede da camada superior, tratando a camada de enlace como uma vasta e opaca nuvem ATM. O ATM é usado como uma tecnologia de transmissão que é acessada através dos serviços oferecidos pela camada de adaptação AAL5.

Esta abordagem leva à ineficiência, complexidade e duplicação de funcionalidades das camadas 2 e 3. A rede ATM possui seu próprio endereçamento e esquema de roteamento completamente independente do IP.

Algumas funcionalidades são requeridas neste tipo de modelo :

> As capacidades de *broadcast* e *multicast* provenientes da natureza sem conexão do IP precisam ser emuladas.
>
> Deve haver um mecanismo de mapeamento de endereços IP ou MAC (*Medium Access Control*) em endereços ATM.

Entretanto, esta abordagem foi amplamente usada devido à sua simplicidade sob o ponto de vista de projeto. Uma comparação entre os modelos do IP sobre ATM desta abordagem é apresentado em [25] onde são chamados de Modelos Clássicos.

Uma outra abordagem é a dos protocolos que propõem o roteamento integrado da camada 2 e da camada 3 em um único protocolo. A integração é obtida concentrando-se toda a função de roteamento em somente uma das duas camadas.

Esta abordagem engloba as propostas proprietárias como o Tag Switching [36] da Cisco, o IpSwithing [14] da Ipsilon, o ARIS [13] (*Aggregate Route Based Ip Switching*) da IBM e o CSR [19][34] (*Cell Switched Router*) da Toshiba.

Além disso, o MPLS [15][30] (*Mutiprotocol Label Switching*) está sendo criado pelo *MPLS Working Group* para padronizar uma tecnologia que integre a técnica de *label-swapping* com o roteamento de camada 3. No MPLS o roteamento é feito somente pelo IP padrão e é traduzido para o ATM através de um protocolo de distribuição de *labels* que substitui a sinalização ATM definida pelo ATM Forum UNI/NNI [1][2][4] (*User-/Network-to-Network Interface*). O resultado é que a sinalização e o roteamento ATM não são necessários para que o ATM ofereça suporte para o tráfego IP [12].

Este capítulo apresenta na seção 2.2 alguns conceitos utilizados pelas propostas citadas acima. A seção 2.3 apresenta uma breve descrição das propostas já apresentadas pelo IETF e pelo ATM Forum para o transporte IP sobre ATM e as seções 2.4 e 2.5 apresentam os fundamentos e o modo de operação dos protocolos *IpSwitching* e *MPLS*, que foram especificados e simulados neste trabalho nos capítulos 3 e 4.

2.2 Técnicas do Roteamento *Shortcut*, *Label Swapping* e Redes NBMA

Devido aos problemas do aumento de tráfego na rede Internet, os roteadores tornaram-se pontos de congestionamento na rede. O objetivo para tentar aliviar a carga e o processamento nos roteadores é transportar mais tráfego de camada 3 diretamente sobre a camada 2. Esta técnica é conhecida como *shortcut routing*.

A definição de *shortcut routing* segundo [12], refere-se aà habilidade de transportar pacotes de camada de rede (camada 3) diretamente sobre a camada de enlace (camada 2). A técnica de *shortcut* é aplicável particularmente à redes NBMA (*Non Broadcast Multiple Acess*).

Uma rede NBMA [8] é uma tecnologia de rede que permite que vários dispositivos sejam acoplados à mesma rede, entretanto não permite o uso do mecanismo de *broadcast*. Uma rede NBMA consiste de um conjunto de nós conectados ponto a ponto, e que não estão fisicamente, ou administrativamente, restringidos de comunicarem-se diretamente.

Utilizar a técnica de *shortcut* no IP sobre o ATM significa obter um misto de transporte orientado a conexão e não orientado a conexão. Conexões de camada 2 podem ser estabelecidas, mesmo que sobre um tráfego puramente sem conexão e de transporte *best-effort*.

Já a técnica de *Label Swapping* faz com que pacotes IP sejam associados a um *label*. O conceito de *Label Swapping* segundo [21] é uma técnica que permite a transmissão dinâmica de dados usando *labels* para identificar classes de pacotes de dados que são tratados indistintamente quando transportados.

Um *label* é um identificador de tamanho curto e fixo, que é usado para identificar um endereço IP, ou um conjunto de endereços.

2.3 Propostas para o Transporte do IP sobre o ATM

Esta seção apresenta uma descrição breve dos protocolos IP Clássico e NHRP propostos pelo IETF e, dos protocolos LANE e MPOA propostos pelo ATM Forum. Além disso apresenta-se a descrição das propostas proprietárias Tag Switching da Cisco e do ARIS da IBM.

2.3.1 IP Clássico

O IP Clássico, (CIP, *Classical IP*) [22], foi a primeira implementação do IETF para o IP sobre o ATM para redes NBMA. O IP Clássico se utiliza do roteamento em camadas (*Layered Routing*). A rede ATM possui seu próprio endereçamento e esquema de roteamento, completamente independente do IP. Assim, o roteamento na camada superior é determinado por protocolos de roteamento IP, como o OSPF (*Open Shortest Path First*) por exemplo, mas a rota entre os roteadores IP é estabelecida sobre o controle de

um esquema de roteamento de camada ATM, como o PNNI [5] (*Private Network Node Interface*).

No IP Clássico, o roteamento IP *hop-by-hop* é diretamente mapeado no ATM, por um mecanismo de resolução de endereço. Este mecanismo utiliza um servidor ATMARP (*ATM Address Resolution Protocol*) em cada uma das LISs (*Logical IP Subnet*). Os membros de cada LIS compartilham o mesmo endereço de subrede IP, mesmo MTU (*Maximum Transmission Unit*) e, mesmo encapsulamento LLC/SNAP (*Logical Link Control/SubNetwork Attachment Point*).

Quando um *host* deseja se comunicar com outro *host* pertencente à mesma LIS, ele adquire o endereço ATM do IP destino, através de um requerimento enviado ao servidor ATMARP. Após adquirir o endereço ATM do destino, o *host* pode estabelecer uma conexão SVC (*Switched Virtual Connection*).

Entretanto, se o *host* destino não pertencer à mesma LIS, o tráfego IP precisa ser roteado *hop-by-hop* através de um roteador que pertença às duas LISs.

2.3.2 NHRP (*Next Hop Resolution Protocol*)

Para resolver o problema de comunicação entre membros NBMA pertencente a duas LISs distintas, isto é, o problema de não poder estabelecer um SVC entre estes membros, foi proposto o protocolo NHRP [18] (*Next Hop Resolution Protocol*).

O NHRP é uma extensão do ATMARP para prover resolução de endereços entre LISs. Ele possui um modelo cliente servidor onde os NHCs (*NHRP Client*) são os clientes, e os NHSs (*NHRP Server*) são os servidores.

Os NHCs são configurados com o endereço ATM de seu servidor e registram seu endereço IP e ATM com o NHS. O NHC envia um pedido de resolução de endereço para o NHS quando deseja se comunicar com outro NHC.

O NHS propaga o pedido *hop-by-hop* em direção ao endereço IP destino. Ao longo do caminho, o NHS que tiver a resolução de endereço para o endereço IP destino, responde com o endereço ATM.

Ao receber o endereço ATM, o NHC pode estabelecer um SVC diretamente com o destino e começar a transmitir os pacotes.

2.3.3 LANE (*Lan Emulation*)

Para acelerar o emprego do ATM, o ATM Forum desenvolveu o LANE [3][6], cuja função é emular uma rede local *Ethernet* IEEE 802.3 ou *Token Ring* IEEE 802.5 no topo de uma rede ATM. O LANE não altera a camada superior que pode operar normalmente.

A função básica do protocolo LANE é resolver endereços MAC(*Medium Acess Control*) em endereços ATM, então os nós da rede podem estabelecer conexões diretas entre si e transmitir dados.

As entidades definidas pelo LANE são :

LECS = *LAN Emulation Configuration Server*,

LES = *LAN Emulation Server*,

BUS = *Broadcast and Unknown Server* e,

LEC = *Lan Emulation Client.*

Cada LEC sempre obtém o endereço ATM do LECS de alguma das formas : ou quando é configurado através do ILMI (*Integrated Layer Management Interface*), utilizando um endereço *anycast* para criar uma conexão com o LECS ou o LEC é configurado com um PVC (*Permanent Virtual Connection*) com o LECS. Desta forma, o LEC obtém o endereço do LES através do LECS e se registra com o seu endereço MAC e ATM no LES.

Quando um LEC quer resolver um endereço, ele envia ao LES uma mensagem *LE_ARP* com o endereço MAC do destino. Se o nó destino estiver registrado, o LES responde com o endereço ATM do destino e o LEC pode então estabelecer uma conexão direta com o destino.

Se o LES não conseguir resolver o endereço, o LEC que deseja a resolução de endereço envia uma mensagem para o BUS que propaga a mensagem para todos os clientes através do *broadcast.*

Da mesma forma que o IP Clássico, dois nós em diferentes LANs Emuladas têm que se comunicar via roteador. O roteador acaba tornando-se um ponto de falha na rede.

2.3.4 MPOA (*MultiProtocol Over ATM*)

O MPOA [7] foi desenvolvido pelo ATM Forum para resolver o problema de que os pacotes tinham que ser transmitidos em nível de camada 3 por um roteador, quando houvesse a comunicação de dois *hosts* que estivessem em diferentes LANs Emuladas, mesmo que sobre a mesma infraestrutura ATM.

O MPOA integra o LANE, preservando os benefícios de *bridging* e o NHRP, para que conexões diretas inter-subredes possam ser estabelecidas sem o requerimento de roteadores (*shortcut routing*). Assim, o MPOA utiliza informações de *bridging* e de roteamento para encontrar o caminho ótimo até o destino.

Além disso, o MPOA utiliza a técnica de *Virtual Routing* que permite a separação física do cálculo de rota de camada de rede, do transporte de dados.

2.3.5 Tag Switching

O *Tag Switching* [36] foi desenvolvido pela Cisco Systems Inc. e utiliza a tecnologia de *label* (aqui chamado de tag) *switching*, para transportar pacotes de camada 3.

O *Tag Switching* é constituído de componentes de transmissão e componentes de controle. Os componentes de transmissão são :

- tags : *labels* carregados pelos pacotes,

- TIB (*Tag Information Base*) : tabela onde cada entrada é um tag de entrada (*incomming tag*) com uma ou mais subentradas, como tag de saída (*outgoing tag*), interface de saída e informações do nível do enlace de saída,

- Procedimento de Transmissão : quando um tag-switch recebe um pacote com um tag, ele usa este tag como um índice na TIB. Encontrando uma entrada com o tag de entrada igual ao tag do pacote, para cada subentrada, o switch substitui o tag e a informação do enlace do pacote, pelo tag de saída e pela informação do enlace de saída e então, envia o pacote na interface de saída.

No *Tag switching* é feito um mapeamento entre um tag e o roteamento de camada de rede. Um tag pode ser associado a um fluxo de aplicação, a uma única rota, a um grupo de rotas ou a uma árvore *multicast*. A principal função do componente de controle é criar as associações de tags e distribuí-las aos *tag-switches* interconectados.

Um comutador de tags (*tag switch*) deve participar nos protocolos de roteamento e construir sua FIB (*Forwarding Information Base*) usando as informações obtidas destes protocolos. O comutador de tags utiliza então um dos três métodos para alocar tags, criar a TIB e distribuir a associação de tags :

- *DownStream Tag Allocation* - o comutador *downstream* toma a decisão de alocar um tag e associá-lo a uma rota particular ou a um conjunto de rotas (um endereço prefixo na FIB),

- *DownStream Tag Allocation on Demand* - o comutador *downstream* toma a decisão de alocar tags e associá-los com um endereço prefixo, mas somente quando ele recebe um pedido de um comutador *upstream*,

- *UpStream Tag Allocation* - o comutador *upstream* toma a decisão de alocar um tag e associá-lo a um endereço prefixo.

A distribuição de tags é realizada utilizando-se o protocolo de distribuição de tags TDP (*Tag Distribution Protocol*). Quando um pacote entra em uma rede de comutadores de tags, um tag é adicionado ao cabeçalho do pacote pelo primeiro comutador de tags da rota. Então o pacote é comutado pela rede e o tag é removido pelo último comutador de tags da rota.

O tag switching é um protocolo orientado a topologia (*topology driven*) pois a existência de uma entrada da FIB causa a alocação de tags.

2.3.6 ARIS (*Aggregate Route Based IP Switching*)

O ARIS [13] foi criado pela IBM. O ARIS usa ISRs (*Integrated Switched Routers*), que são comutadores que têm suporte para o roteamento de camada de rede e caminhos comutados pré-estabelecidos para nós de saída bem conhecidos.

Os identificadores dos nós de saída (*egress*) definem um caminho roteado na rede. Eles são classificados em diferentes tipos de acordo com a informação do protocolo de roteamento. Um exemplo de identificador seria o *Egress IP Adress* que é usado no protocolo de roteamento OSPF (*Open Shortest Path First*).

Os caminhos comutados, na forma de árvores roteadas para os pontos de saída, são estabelecidos pelos ISRs através da troca de mensagens ARIS [13].

Ao receber um pacote, o ISR de entrada (nó de entrada da rede) executa um procedimento de procura na sua FIB (*forwarding information base*), da maior combinação de endereço (*longest match*), e obtém o label associado ao caminho comutado ao destino. Então o pacote é transmitido no caminho comutado. Se não for encontrada nenhuma entrada na FIB, o pacote é transmitido para o próximo *hop* através da trasmissão *hop-by-hop*.

2.4 IpSwitching

O IpSwitching [27] é uma alternativa para a integração IP e ATM. Esta tecnologia foi desenvolvida pela Ipsilon Networks Inc. e propõe o uso do IP diretamente sobre o topo do hardware ATM, descartando a necessidade de um protocolo de sinalização ou, um protocolo de resolução de endereços. O objetivo do IpSwitching é exatamente tomar vantagem das características de comutação rápida do hardware ATM, mas também tomar proveito das características de flexibilidade da natureza sem conexão do IP.

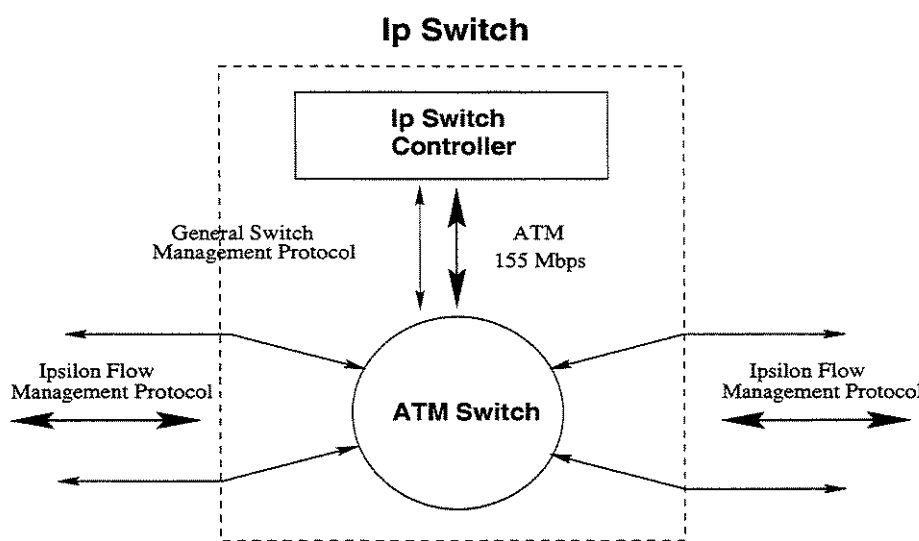

Figura 2.1: Estrutura de um Ip Switch [27]

Um IpSwitch é construído tomando-se um comutador ATM padrão sem qualquer modificação. Entretanto, todo o software residente no processador de controle acima da camada de adaptação AAL-5 é removido. A modificação do software de controle faz com que cada IpSwitch atue de maneira sem conexão [26].

Desta forma, um IpSwitch figura 2.1, é definido pelo hardware ATM e por um controlador do IpSwitch. O controlador do IpSwitch é um processador que roda um software de roteamento IP padrão com extensões como o IFMP (*Ipsilon Flow Management Protocol*), um classificador de fluxo e o GSMP (*Generic Switch Management Protocol*).

O IFMP [14] é o protocolo responsável por associar os fluxos IP a canais virtuais ATM.

O classificador de fluxo, através de uma política local, tem por função a tomada de decisão sobre comutar ou não cada fluxo IP. E o GSMP [28] é um protocolo de controle simples e de baixo nível que é rodado no lugar do software ATM, e tem a função de controlar o hardware do comutador ATM.

2.4.1 Ipsilon Flow Management Protocol (IFMP)

O IFMP é um protocolo que permite que um nó possa advertir um nó adjacente, a associar ou liberar um *label* de camada 2 a um fluxo IP. O formato do *label* depende do tipo de enlace físico no qual ele será transmitido. Um *label* para a transmissão de datagramas IPv4 sobre ATM é transmitido através dos elementos de mensagens do IFMP. Seu formato é definido de acordo com a figura 2.2 [29]. Os 16 bits de mais baixa ordem correspondem ao VCI (*virtual channel identifier*) e os 12 bits seguintes de mais alta ordem, correspondem ao VPI (*virtual path identifier*). Os quatro bits mais significantes do *label* são reservados.

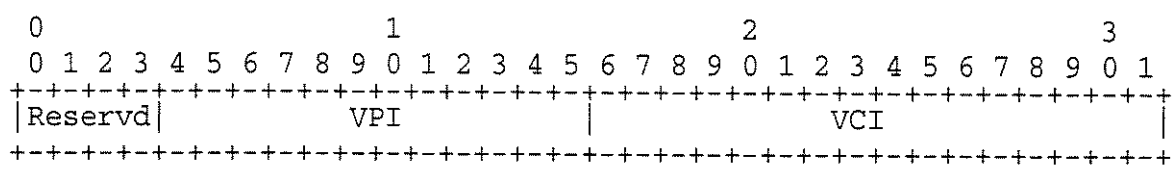

Figura 2.2: *Label* IPv4 em enlaces ATM [29]

O funcionamento do IFMP, após o classificador de fluxo do controlador de um switch IP ter decidido por comutar um fluxo, é descrito nos passos a seguir de acordo com a figura 2.3.

1. O controlador do Ip Switch seleciona um *label* livre x na porta de entrada i pelo qual o pacote foi recebido. Seleciona um *label* livre x' na sua porta de controle c, e instrui o *driver* do comutador a mapear o *label* x na porta de entrada, para o *label* x'na porta de

controle c.

2. Uma mensagem IFMP (contendo o identificador de fluxo, o *label* e o *lifetime*) é enviada para o nó acima, ou seja para o nó do qual o pacote veio, requerendo que este nó envie todos os próximos pacotes deste fluxo, em um canal virtual ATM especificado pelo *label* da mensagem. Deste ponto em diante, os pacotes chegam no controlador do IpSwitch com o *label* x'. Embora os pacotes ainda sejam remontados, o processo é acelerado porque a decisão de roteamento anterior, já foi armazenada.

3. O IpSwitch pode receber pedidos de redirecionamento similares na porta j do roteador do próximo nó, para redirecionar o fluxo no *label* y. A decisão para aceitar este requerimento é local e pode ser ignorada.

4. Aceitando o pedido de redirecionamento, o IpSwitch mapeia o *label* x na porta de entrada i, para o *label* y na porta de saída j. Deste ponto em diante, todos os pacotes deste fluxo são comutados no hardware e não remontados como era feito anteriormente.

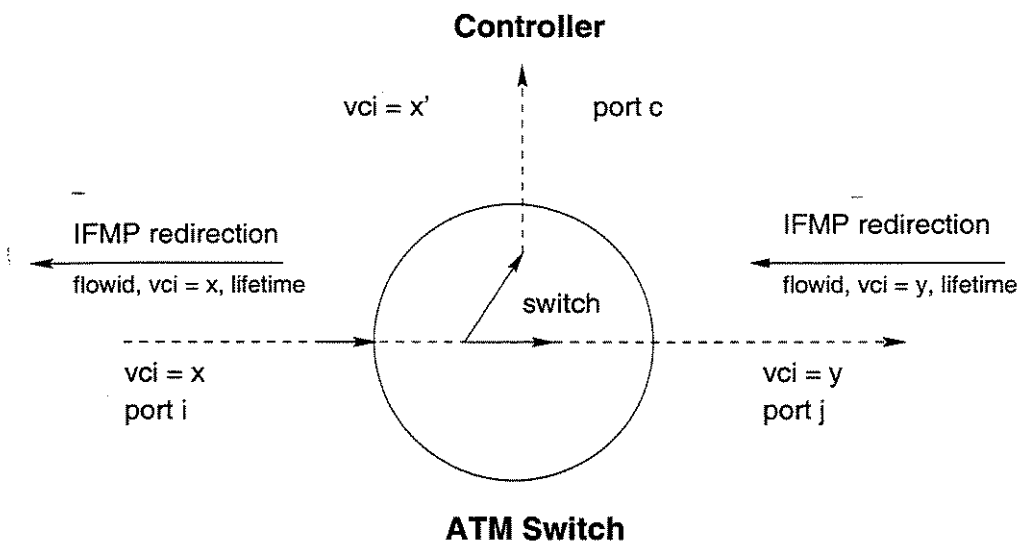

Figura 2.3: Estabelecimento de um Fluxo Comutado pelo IpSwitching [27]

2.4.2 Classificação de Fluxo

O conceito de fluxo surgiu dentro da comunidade IP e é definido como sendo uma sequência de pacotes que são enviados de uma origem particular para um destino particular (*unicast ou multicast*), e que são tratados da mesma forma pelas funções de roteamento [14].

O classificador de fluxo é uma entidade que não é especificada pelo IFMP, mas que deve estar presente em todos os IpSwitches, e que é responsável por decidir se um fluxo

deve ou não ter um VC ATM dedicado ao seu transporte.

Para atender aos objetivos do IpSwitching, o classificador de fluxo deve ser projetado de forma a comutar o maior número possível de pacotes em nível de camada 2, procurando não gastar grande quantidade de espaço de VCs do comutador ATM e também, evitar o envio de mensagens redundantes de redirecionamento do IFMP. De certa forma, espera-se que o classificador de fluxo tenha efeito na performance do IpSwitching.

Devido ao requerimento de espaço de VCs ATM, procura-se classificar os fluxos que possivelmente terão uma longa duração para comutação e continuar o transporte de pacotes pelo caminho de roteamento padrão do IP dos fluxos de curta duração. O ideal, isto é, uma performance de quase 100%, seria comutar todos os fluxos, tanto os de longa duração como os de curta duração, porém um espaço de VCs ATM consideravelmente grande seria necessário.

Os fluxos de longa duração são os fluxos que apresentam grande quantidade de tráfego para ser comutado. São considerados fluxos de longa duração os fluxos que apresentam uma boa probabilidade de tráfego intenso durante um determinado tempo como tráfego multimídia, voz, imagem e vídeo conferência por exemplo. Outros bons candidatos a comutação são os fluxos que transportam tráfego de tempo real e que tenham requisitos de qualidade de serviço.

Já os fluxos de curta duração como DNS (*Domain Name Server*) *queries*, breves transações entre cliente e servidor, pesquisa em banco de dados, etc, continuam sendo transmitidos pelo transporte de pacotes *hop-by-hop* entre roteadores IP usando conexões compartilhadas estabelecidas entre os roteadores.

Há três tipos de classificadores de fluxo que preocupam-se com as características do fluxo para a tomada de decisão, o classificador X/Y, o de protocolo e o de porta.

O classificador X/Y classifica um fluxo como candidato a comutação se em Y segundos forem transmitidos X pacotes, todos pertencentes ao mesmo fluxo.

O classificador de protocolo associa todos os pacotes TCP a fluxos. Apesar do IP ser um protocolo não orientado a conexão, são empregadas sobre o IP aplicações com protocolo de transporte orientado a conexão.

Finalmente, o classificador de porta usa o número da porta de camada de transporte para decidir quais aplicações têm fluxos de maiores durações e devem ser comutados.

Como o tipo de classificador de fluxo pode interferir no desempenho do protocolo IpSwitching, ele foi escolhido como objeto de estudo no capítulo 3. **Um classificador de fluxo do tipo X/Y foi especificado e simulado. O objetivo do trabalho apresentado no capítulo 3, após a especificação formal do IpSwitching, é analisar através de simulações com *traces* reais de tráfego IP, o desempenho do protocolo IpSwitching em relação à mudança dos parâmetros X e Y do classificador de fluxo.**

2.4.3 Tipos de Fluxos

Um fluxo é identificado pelo seu identificador de fluxo. Diferentes tipos de fluxos podem ser definidos de acordo com um conjunto particular de campos do cabeçalho do

pacote IP, usados para identificar um fluxo e que devem ser invariantes para todos os pacotes pertencentes ao mesmo fluxo em qualquer ponto da rede [14].

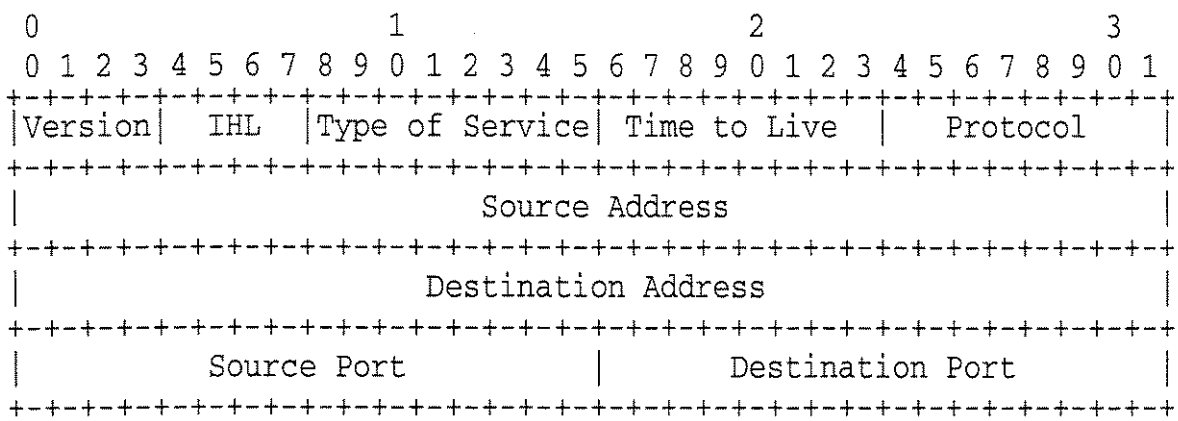

Figura 2.4: Identificador de Fluxo do Tipo 1 [14]

Um fluxo IP é caracterizado de acordo com os campos do cabeçalho IP/TCP/UDP como : tipo de serviço, protocolo, endereço origem, endereço destino, porta de origem, porta de destino, etc, e que determinam a decisão de roteamento. Dois pacotes pertencem ao mesmo fluxo se os valores destes campos são idênticos [27].

Três tipos de fluxos são definidos na versão 1.0 do IFMP : tipo de fluxo 0, tipo de fluxo 1 e tipo de fluxo 2. O tipo de fluxo 0 é usado para mudar o encapsulamento padrão dos pacotes do IPv4. O identificador de fluxo para o tipo de fluxo 0 é nulo e o *Flow ID Length* é igual a zero.

O tipo de fluxo 1 é designado para protocolos como o UDP e TCP no qual os quatro primeiros octetos após o cabeçalho IPv4 especificam o número da porta origem e o número da porta destino. O *Flow ID Length* para o fluxo do tipo 1 é igual a 4 palavras de 32 bits e o formato do identificador de fluxo é mostrado na figura 2.4.

O tipo de fluxo 2 possui um *Flow ID Length* de 3 palavras de 32 bits e o formato de seu identificador de fluxo é mostrado na figura 2.5.

O tipo de fluxo 2 foi adotado na especificação do capítulo 3 por ser o tipo de fluxo que proporciona um maior desempenho do IpSwitching, já que um fluxo do tipo 2 contém vários fluxos do tipo 1.

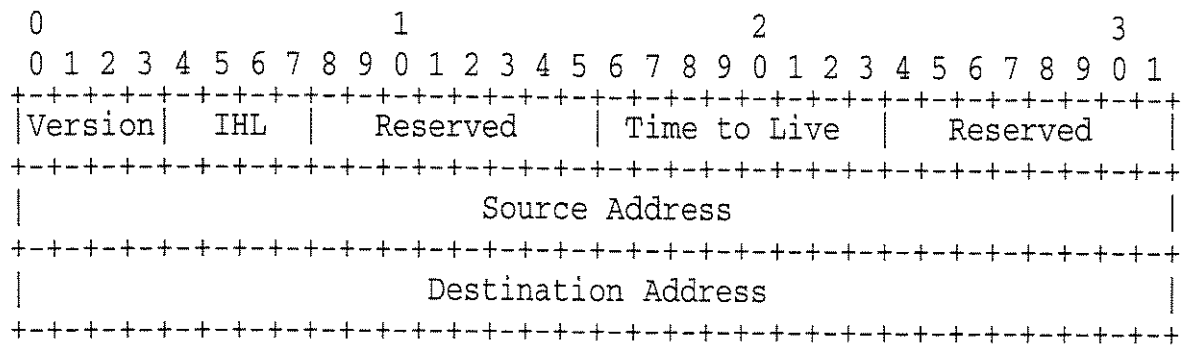

Figura 2.5: Identificador de Fluxo do Tipo 2 [14]

2.5 MPLS (*Multiprotocol Label Switching*)

2.5.1 Uma Visão Geral do MPLS

O MPLS [15][30] foi desenvolvido pelo MPLS *Working Group* com o objetivo de se padronizar uma tecnologia que integrasse o paradigma de *label swapping* com o roteamento de camada de rede.

No roteamento *hop-by-hop* dos protocolos de rede sem conexão como o IP, a decisão de roteamento é tomada em cada *hop* conforme o pacote vai atravessando a rede. Cada nó roteador analisa o cabeçalho do pacote e roda um algoritmo de roteamento para calcular o próximo *hop*.

A escolha do próximo *hop* baseia-se em duas funções : particionar todo o conjunto de possíveis pacotes em FECs (*Forwarding Equivalence Classes*) e associar a cada FEC um próximo *hop*.

No tranporte convencional IP , conforme um pacote atravessa a rede, uma decisão independente de transporte é realizada em cada ponto da rede. Cada roteador roda um algoritmo de roteamento de camada de rede, e com a análise do cabeçalho do pacote e com os resultados do algoritmo de roteamento é tomada a decisão de qual será o próximo *hop* para o pacote.

A escolha de um próximo *hop* pode ser considerada como na composição de duas funções. A primeira função é particionar o conjunto total dos possíveis pacotes em classes de transmissão equivalentes, chamadas FECs (*Forwarding Equivalence Classes*). A segunda função é mapear cada FEC a um próximo *hop*. Assim, conforme as decisões de transmissão são tomadas, diferentes pacotes que são mapeados na mesma FEC, tornam-se indistinguíveis, isto é, pacotes que pertencem a uma FEC seguirão o mesmo caminho na rede.

No roteamento convencional IP um roteador considera que dois pacotes pertencem à mesma FEC, se existe um endereço prefixo X nas tabelas de roteamento do roteador, sendo que X é o maior prefixo e que combina com os endereços destino dos dois pacotes. Assim, pacotes que são associados a mesma FEC seguem o mesmo caminho e são indistinguíveis. A cada *hop* o cabeçalho do pacote é reexaminado e associado a uma FEC.

Uma das vantagens do MPLS segundo [15] é que a análise do cabeçalho de camada de rede e a associação do pacote a uma FEC são realizadas somente uma vez, quando o pacote entra na rede (no nó MPLS de entrada). Cada FEC é associada a um *label* de tamanho fixo e pequeno. Como os pacotes pertencentes à mesma FEC são transportados da mesma maneira, eles recebem o *label* associado a FEC ao qual eles pertencem.

Quando o pacote é passado ao próximo *hop* com o *label*, o próximo *hop* não precisa rodar o algoritmo de roteamento e analisar o cabeçalho de camada de rede do pacote. O *label* enviado com o pacote (*label de entrada*) para o próximo *hop*, serve como um índice de entrada para uma tabela que indica o novo *label* (*label de saída*) que o pacote deve receber para ser transmitido para o próximo *hop*.

Assim os pacotes vão sendo transmitidos através da rede sem que seja executado, em cada *hop* da rede, a análise de cabeçalho para a escolha do próximo *hop*.

O MPLS , através da técnica chamada *label swapping*, permite um paradigma de transporte mais simples que as redes baseadas no transporte *hop-by-hop* como o IP convencional, além de facilitar a construção de roteadores de alta velocidade. Um nó transmitindo pacotes em nível de camada de rede deve realizar uma análise de um cabeçalho bem mais longo, e executar um processo de combinação do maior prefixo para determinar o caminho de transmissão. A transmissão em nível de camada de enlace provê uma transmissão mais simples e mais rápida devido ao uso de *labels* de tamanho fixo e curto.

Para associar *labels* às FECs, o MPLS utiliza um protocolo para distribuir e manter *labels* chamado LDP (*Label Distribution Protocol*) [21]. **O LDP é objeto de estudo deste trabalho e, seus procedimentos de distribuição de *labels* foram especificados e simulados no capítulo 4.**

As subseções a seguir apresentam uma descrição dos conceitos básicos adotados pelo MPLS, assim como o modo de operação do LDP.

2.5.2 *Labels*

Um *label* é um identificador de tamanho fixo e pequeno usado no MPLS para identificar uma FEC.

Um pacote é dito rotulado (*labeled*) quando possui um *label* representando a FEC ao qual o pacote pertence.

O *label* é codificado em um campo que esteja disponível nos cabeçalhos da camada de enlace ou da camada de rede, ou utilizando-se um encapsulamento do pacote com um cabeçalho próprio para o propósito.

2.5.3 FEC (*Forwarding Equivalence Class*)

Uma FEC é definida como um grupo de pacotes de camada 3 que são transportados da mesma maneira, isto é, pelo mesmo caminho e com o mesmo tratamento de transporte.

Uma FEC é formada por um ou mais elementos FEC, onde cada elemento FEC especifica quais pacotes devem ser associados a cada LSP (*Label Switched Path*). Se o LSP

possui mais de um elemento FEC, ele deve terminar no nó, ou antes do nó, onde os elementos FEC não podem mais compartilhar o mesmo caminho.

Dois tipos de elementos FEC estão definidos em [21] :

- Prefixo de Endereço IP : um prefixo de endereço IP de qualquer tamanho entre 0 e 32 bits,

- Endereço Host : um endereço IP de 32 bits.

Um pacote IP é associado a um LSP, se o LSP possui um elemento FEC de prefixo de endereço IP que combina com o endereço IP destino do pacote, ou seja, o endereço destino IP começa com o prefixo de endereço IP (*matching prefix*).

As regras para se associar um pacote a um LSP são :

- o pacote é associado ao LSP, se o LSP possui um elemento FEC de endereço *host* idêntico ao endereço IP destino do pacote,

- se existir mais de um LSP com elemento de FEC de prefixo IP que combine com o endereço destino do pacote IP, o pacote é associado ao LSP que possuir o elemento que ocorra a combinação mais longa (*longest match*),

- se o LSP possuir um elemento FEC que contenha o endereço (de 32 bits) de um roteador, e sabe-se que o pacote passará por este roteador, o pacote é associado ao LSP.

2.5.4 *Label Switching Routers (LSRs) Upstream e Downstream*

Um nó MPLS é definido como um nó que roda o MPLS [30]. O nó MPLS deve estar ciente dos protocolos de controle MPLS, e operar com um ou mais protocolos de roteamento de camada 3, além de ser capaz de transportar pacotes baseados em *labels*.

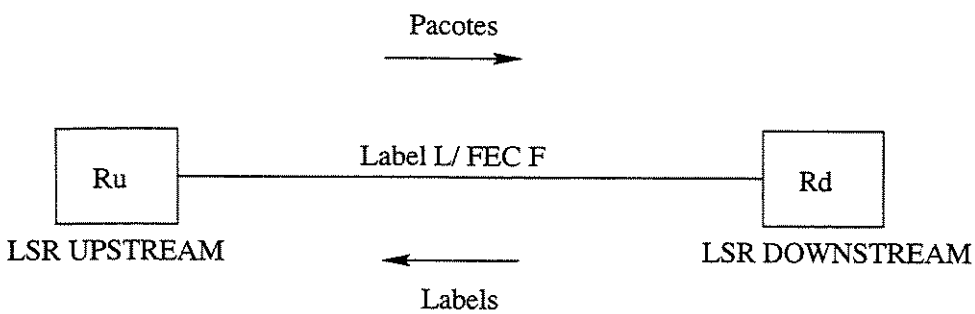

Figura 2.6: Distribuição de Labels do LDP

Um LSR, (*Label Switching Router*), é um nó MPLS capaz de transportar pacotes nativos de camada 3.

De acordo com a figura 2.6, dado um LSR Ru e um LSR Rd que tenham combinado de associar o *label* L para a FEC F, e os pacotes vão de Ru para Rd, então o LSR Ru é o LSR *Upstream* e o LSR Rd é o LSR *Downstream* em relação a esta associação.

A associação de *labels* em MPLS é realizada pelo LSR *Downstream*, *DownStream-Assigned* ou *Downstream Label Allocation*, isto é, pelo LSR que usa o *label* como um índice para suas tabelas de comutação. O LSR cria seus *labels* de acordo com a necessidade e conforme as informações de suas tabelas de roteamento e, então distribui os *labels* a todos os *LDP Peers*.

A figura 2.6 simboliza o sentido da transmissão dos pacotes e da associação de *labels* entre dois LDP *Peers*, o LSR Ru (*LSR UpStream*) e, o LSR Rd (*LSR DownStream*).

2.5.5 Alocação de Labels Upstream

Na alocação de *labels* upstream, a alocação de *labels* é realizada pelo LSR *Upstream*.

Este tipo de alocação vem como uma possibilidade para tentar otimizar a máquina de *multicast*, pois um LSR poderia escolher o mesmo *label* para todas as portas de saída no qual um pacote *multicast* fosse destinado.

2.5.6 Distribuição de *Labels DownStream* x *DownStream On Demand*

A distribuição de *labels* no MPLS se dá na direção *downstream to upstream*. O MPLS possui dois tipos de alocação de *labels*, a distribuição de *labels DownStream* e, a distribuição de *labels Downstream On Demand*.

Na distribuição de *labels Downstream On Demand* um LSR pode requerer explicitamente do seu próximo *hop*, por uma associação de *label* para uma determinada FEC. Isto significa que, um LSR só irá requisitar um *label* de um LDP *peer*, quando seus cálculos de roteamento, indicarem que este *peer* é o próximo *hop* para a rota. Este tipo de alocação de *labels* é utilizada principalmente em casos particulares como o ATM que pode possui um número limitado de *labels* que podem ser usados através de uma interface, ou um número limitado de associação de *labels* suportado por um dispositivo.

Já na distribuição de *label DownStream*, um LSR pode distribuir associação de *labels* a LSRs mesmo que estes não os tenham requisitado explicitamente.

2.5.7 Controle LSP Independente e Ordenado

O estabelecimento de LSP pode ser feito de duas maneiras no LDP : pelo Controle LSP Independente ou pelo Controle LSP Ordenado [15].

No controle LSP Independente, o LSR toma uma decisão independente sobre associar um *label* a uma FEC e distribuí-lo aos seus LDP *peers*. Ele pode enviar *label mappings* para os seus *peers* vizinhos em qualquer instante que desejar. Se o LSR estiver operando por exemplo, no modo de distribuição de label *DownStream On Demand* Independente,

quando o LSR receber um *request* de um *peer upstream*, ele já pode responder com o *label upstream*, antes mesmo que receba um *label* do *peer downstream* que é o próximo *hop* para a FEC. Se o LSR estiver operando no modo de distribuição *DownStream* Independente, o LSR ele pode enviar *label mappings* para os seus *peers* vizinhos, a qualquer instante que estiver preparado para realizar a comutação de labels para a FEC.

Já no controle LSP Ordenado, o LSR só associa um *label* a uma FEC se ele for o LSR nó de saída da rede para a FEC, ou se ele já tiver recebido um mapeamento de *label* do seu próximo *hop* para a FEC. Caso o LSR não seja nó de saída da rede e não tenha recebido um *label* do próximo *hop*, ele propaga o *request* para o próximo *hop* e aguarda pelo envio do mapeamento. Somente quando o LSR receber o *label* do Next-Hop, ele alocará um *label upstream* disponível e, o propagará para o *peer upstream* que enviou o *request*.

O modelo de controle LSP Ordenado deve ser usado, quando se deseja assegurar que um conjunto de propriedades vai ser atendida por todo o LSP. Como no modelo independente o LSR já aloca *labels upstream* e os distribui a seus *peers* sem ter recebido o mapeamento de *label* do seu próximo *hop downstream*, o tráfego pode começar a ser comutado nos LSRs antes que todo o LSP tenha sido estabelecido. Desta forma, as propriedades podem não ser atendidas em todo o LSP já que, o LSR alocou o *label upstream* sem saber se o próximo *hop* para a FEC, poderá antender as propriedades requeridas.

Um outro problema do modo Independente é que, pode acontecer o envio de pacotes sem *label* para o *peer downstream* se, o label *upstream* foi distribuído, antes que o LSR receba o mapeamento do *label downstream*.

2.5.8 Modo de Rentenção de *Label* Conservativo e Liberal

Foram definidos dois modos de retenção de *label* no MPLS, o modo de renteção de *label* Conservativo e, o modo de retenção de *label* Liberal [15].

O modo de retenção de *label* Liberal, retém as informações de mapeamentos recebidos de todos os LDP *peers*, mesmo que eles não sejam um próximo *hop* para uma das FECs reconhecidas pelo LSR.

Já o modo de retenção de *label* Conservativo, só retém as informações de mapeamentos enviados por *peers* que são considerados *Next-Hops* válidos, isto é, para os *Next-Hops* no qual o LSR enviará pacotes.

Quando o LSR está usando o modo de distribuição *DownStream On Demand* ele irá requisitar mapeamentos de *labels* somente para os *Next-Hops* válidos de acordo com sua tabela de roteamento. Neste caso, o modo de retenção Conservativo é desejado, principalmente porque LSRs que se utilizam do modo *DownStream On Demand* possuem um espaço de *labels* limitado como LSRs ATM.

Já no modo *DownStream*, o LSR receberá mapeamentos de todos os seus LDP *peers*. A decisão de se utilizar o modo de retenção de *label* Conservativo ou Liberal, depende exclusivamente da capacidade do LSR.

A desvantagem de se utilizar o modo Conservativo é que, se houver uma mudança de próximo *hop* para uma FEC, o LSR terá que requisitar um mapeamento do próximo *hop*

antes de começar a transmitir os pacotes.

O modo de retenção Liberal por sua vez tem como desvantagem, o armazenamento de mapeamentos e a distribuição de *label* que talvez não fossem necessários.

2.5.9 LDP (*Label Distribution Protocol*)

O LDP, *Label Distribution Protocol* [21], é um conjunto de procedimentos e mensagens no qual, LSRs informam uns aos outros dos mapeamentos de label/FEC realizados.

Através do LDP, LSRs estabelecem LSPs (*Label Switched Paths*) em uma rede, mapeando informações de roteamento de camada de rede, diretamente em caminhos comutados de camada de enlace.

Um LSP é o caminho criado pela concatenação de um ou mais *label switched hops* permitindo que, um pacote seja transmitido trocando-se *labels* (*label swapping*) de um nó MPLS para outro.

Dois LSRs são ditos *LDP Peers* em relação à informação que trocam, quando usam o LDP para trocar mapeamentos label/FEC. Existe entre *LDP Peers* uma sessão LDP (*LDP Session*). A sessão LDP permite que cada *peer* aprenda o mapeamento de *label* do outro, isto é, o protocolo é bi-direcional.

O LDP associa cada LSP criado a uma FEC. A FEC associada ao LSP especifica quais os pacotes que estão mapeados para trafegar no LSP.

2.5.10 Tipos de Mensagens LDP

Existem quatro categorias de mensagens LDP : as mensagens de descoberta (*Discovery Messages*), as mensagens de sessões (*Session Messages*), as mensagens de aviso (*Advertisement Messages*) e, as mensagens de notificação (*Notification Messages*).

O LDP usa o TCP para o envio das mensagens de sessão, aviso e notificação, por ser um transporte confiável. Somente as mensagens de descoberta são enviadas pelo UDP.

Esta seção apresenta uma descrição breve dos quatro tipos de mensagens e suas funções.

- As mensagens de descoberta, *Discovery Messages*, são usadas por um LSR para anunciar e manter a sua presença em uma rede. As mensagens de descoberta, *Hello Messages*, são enviadas periodicamente e são trasmitidas como pacotes UDP para a porta de endereço de grupo *multicast All Routers*.

O recebimento de uma mensagem de *Hello* identifica uma adjacência com um LDP *peer* no nível de camada de rede (*Hello Adjacency*). Um LSR pode decidir estabelecer uma sessão com um LSR que foi aprendido através das mensagens *Hello*.

- As mensagens de sessão, *Session Messages*, são usadas para estabelecer, manter e terminar uma sessão entre dois LDP *peers*.

O estabelecimento de uma sessão LDP se dá em dois passos. O primeiro passo consiste no estabelecimento de uma conexão de transporte e o segundo passo, consiste na

inicialização de uma sessão.

Na sessão de inicialização, depois de estabelecido uma conexão de transporte TCP entre os LDP *peers*, os LSRs negociam parâmetros de sessão trocando mensagens LDP de inicialização. Os parâmetros negociados são por exemplo, a versão do protocolo LDP, o método de ditribuição de *label*, valores dos *timers*, intervalos VPI/VCI para *labels* ATM, intervalos DLCI para *labels* Frame Relay, etc.

Se a negociação for completada com sucesso, a sessão está estabelecida e, os LDP *Peers* podem trocar mensagens de aviso.

- As mensagens de aviso, *Advertisement Messages*, são usadas para criar, mudar e destruir mapeamentos de *label* para FECs. Este trabalho dá ênfase às mensagens de aviso que são utilizadas para o estabelecimento de LSPs com a associação de *labels* a FECs. Estas mensagens são descritas em 2.5.11, *Mensagens LDP de Distribuição de Labels* e, são **utilizadas para a especificação do comportamento de um LSR no capítulo 4.**

- As mensagens de notificação, *Notification Messages*, podem ser de dois tipos, mensagem de notificação de erro ou, mensagem de notificação de aviso.

As mensagens de notificação de erros são usadas para sinalizar erros fatais. Se uma mensagem de notificação de erro é recebida por um LSR em uma sessão LDP, a conexão de transporte TCP é fechada e, todos os mapeamentos de *label* aprendidos na sessão, são descartados.

As mensagens de notificação de avisos são usadas para passar uma informação do LSR sobre a sessão LDP ou, o status de alguma mensagem previamente recebida pelo *peer*.

Todas as mensagens LDP seguem o formato apresentado na figura 2.7.

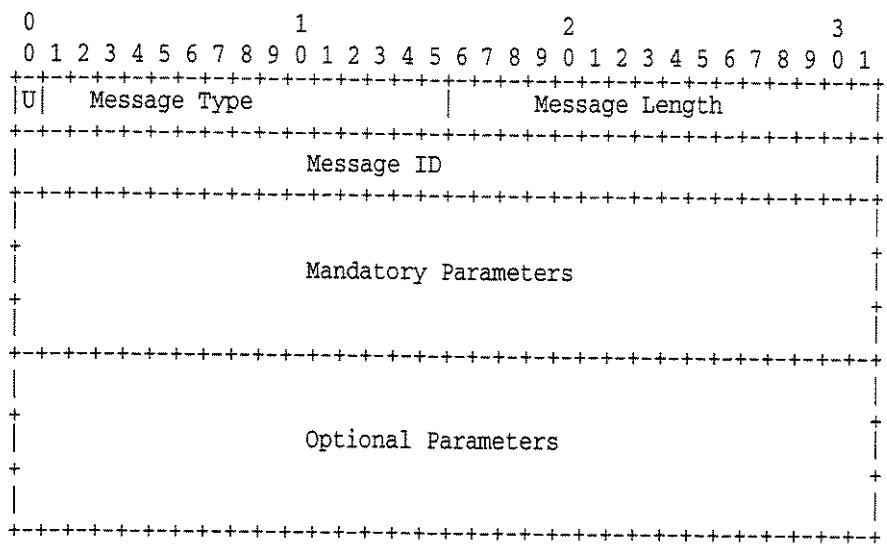

Figura 2.7: Formato das Mensagens do LDP [21]

O campo *U* é o bit usado na ocorrência do recebimento de uma mensagem não conhecida. Se este campo estiver setado para 0, uma mensagem de notificação é enviada ao nó origem, e se estiver setado para 1, a mensagem é ignorada.

O campo *Message Type* identifica o tipo da mensagem.

O campo *Message Length* especifica o tamanho total da mensagem em octetos incluindo *Message ID*, *Mandatory Parameters* e *Optional Parameters*.

O campo *Message ID* é um valor de 32 bits utilizado para identificar a mensagem.

O campo *Mandatory Parameters* conjunto de variáveis de parâmetros requeridos pela mensagem.

E o campo *Optional Parameters* conjunto de variáveis dos parâmetros opcionais da mensagem.

Tanto os parâmetros requeridos como os parâmetros opcionais, seguem o formato TLV (Type, Length and Value) que significa que o tipo, o tamanho e o valor da variável serão apresentados sempre nesta ordem.

2.5.11 Mensagens LDP de Distribuição de *Labels*

Esta subseção apresenta uma descrição das mensagens de aviso, *Advertisement Messages*, que são usadas para criar, mudar e destruir mapeamentos de *labels* à FECs.

As mensagem de aviso são : mensagem de *Label Mapping*, mensagem de *Label Request*, mensagem de *Label Withdraw* e, mensagem de *Label Release*. **Estas quatro mensagens foram utilizadas na especificação do capítulo 4, seção 4.3.2,** *Especificação do Processo StateMachine*, **sendo representadas como sinais.** A ação do LSR diante dos eventos que são simbolizados pelo recebimento de qualquer uma das mensagens de aviso, foram especificados no capítulo 4, segundo os procedimentos de distribuição de *labels* apresentados no apêndice A (*LDP Label Distribution Procedures*), da especificação do LDP [21].

A seguir apresenta-se a descrição da função e o formato de cada uma das mensagens.

- **Mensagem de** *Label Mapping*

A mensagem de *mapping* é enviada por um LSR a um LDP *peer*, para notificá-lo sobre os mapeamentos label/FEC que lhe dizem respeito.

O formato da mensagem de *mapping* é apresentado na figura 2.8. O identificador do tipo da mensagem é o valor 0x0400. A mensagem de *mapping* requer dois parâmetros (*Mandatory Parameters*), a FEC e o *label*.

O campo *FEC TLV* especifica a FEC do *mapping* que está sendo notificado. O campo *Label TLV* especifica o *label* que está sendo associado a FEC do *mapping* notificado. Os parâmetros opcionais são definidos em [21].

- **Mensagem de** *Label Request*

A mensagem de *request* é enviada por um LSR a um LDP *peer* para requerer um mapeamento para uma FEC.

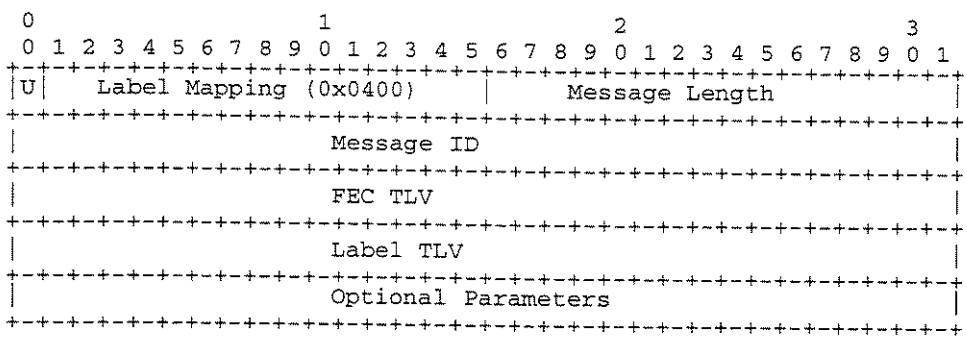

Figura 2.8: Formato da Mensagem de *Mapping* do LDP [21]

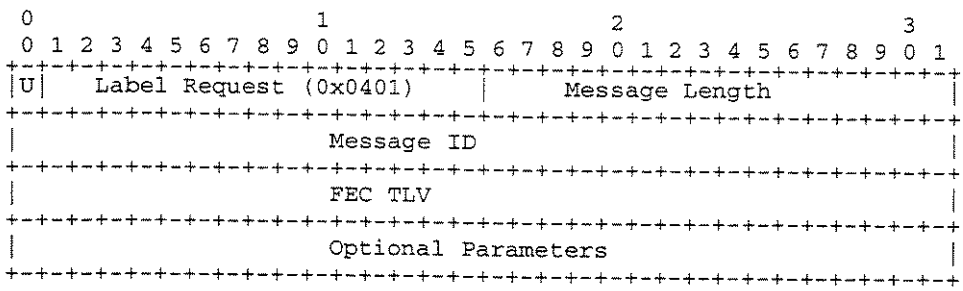

Figura 2.9: Formato da Mensagem de *Request* do LDP [21]

O formato da mensagem de *request* é apresentado na figura 2.9. O identificador do tipo da mensagem é o valor 0x0401. A mensagem de *request* requer apenas um parâmetro (*Mandatory Parameters*), a FEC ao qual se está fazendo um pedido de mapeamento.

O campo *FEC TLV* especifica a FEC do *request* que está sendo enviado. Os parâmetros opcionais são definidos em [21].

- **Mensagem de** *Label Withdraw*

A mensagem de *withdraw* é enviada por um LSR a um LDP *peer*, para sinalizar ao *peer* que, ele não deve continuar a usar o mapeamento FEC-label que o LSR notificou previamente. A mensagem de *withdraw* faz com que a associação FEC-label seja rompida.

O formato da mensagem de *withdraw* é apresentado na figura 2.10. O identificador do tipo da mensagem é o valor 0x0402. A mensagem de *withdraw* requer dois parâmetros, a FEC que é obrigatória e o *label* que é opcional.

O campo *FEC TLV* especifica a FEC no qual o mapeamento FEC/label está sendo destruído (*withdrawn*). O campo *Label TLV*, se presente, especifica o *label* que deve ser retirado da associação FEC/label.

- **Mensagem de** *Label Release*

A mensagem de *release* é enviada por um LSR a um LDP *peer*, para sinalizar ao *peer* que, o LSR não precisa mais do mapeamento FEC/label específico, previamente requerido e/ou notificado pelo *peer*.

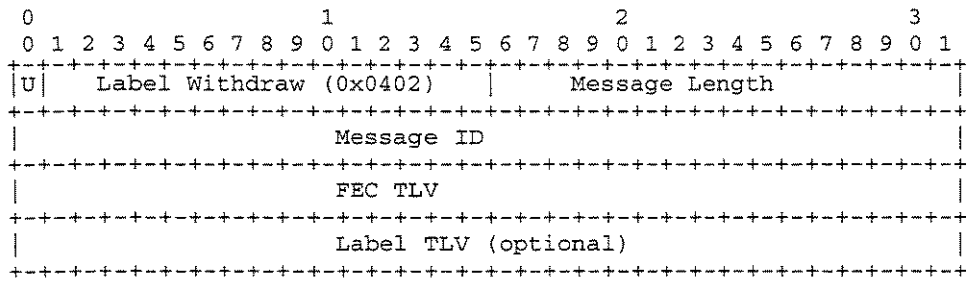

Figura 2.10: Formato da Mensagem de *Withdraw* do LDP [21]

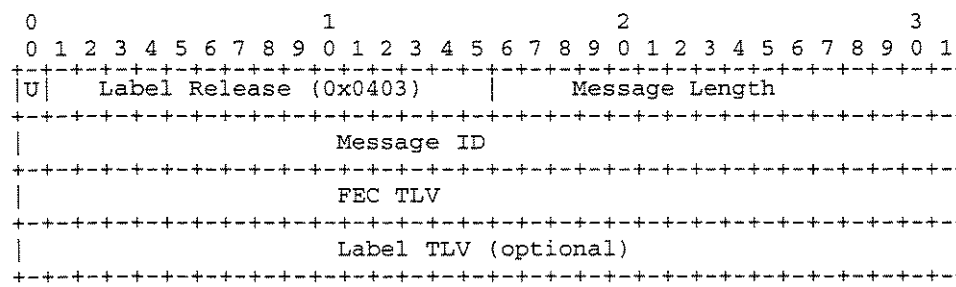

Figura 2.11: Formato da Mensagem de *Release* do LDP [21]

O formato da mensagem de *release* é apresentado na figura 2.11. O identificador do tipo da mensagem é o valor 0x0403. A mensagem de *release* também requer dois parâmetros, a FEC que é obrigatória e o *label* que é opcional.

O campo *FEC TLV* especifica a FEC no qual o mapeamento FEC/label está sendo liberado (*released*). O campo *Label TLV*, se presente, especifica o *label* a ser liberado.

2.5.12 ATM *Label Switching Routers*

Comutadores ATM podem ser usados como LSRs, ATM-LSRs [11], rodando algoritmos de roteamento de camada de rede, como OSPF (*Open Shortest Path First*), e o transporte dos dados baseado nos resultados destes algoritmos.

Para o comutador ATM suportar *label switching* ele deve implementar o componente de controle do *label switching* que, consiste nos procedimentos de alocação, distribuição e manutenção de *labels*. O suporte de *label switching* em um comutador ATM, não requer o componente de controle ATM definido pelo ITU-T e pelo ATM Forum (por exemplo UNI [1][2][4] e PNNI [5]).

Em ATM-LSRs o *label* é transportado no campo VCI/VPI, ou quando dois ATM-LSRs estão conectados através de um *Virtual Path* ATM, o *label* é transportado no campo VCI.

Os ATM-LSRs possuem algumas características de capacidade de hardware e formato imposto pelo ATM para o transporte de células, que interferem no comportamento dos ATM-LSRs. Por exemplo, a função de *label swapping* é executada nos campos VCI/VPI do cabeçalho da célula, restringindo o tamanho e localização dos *labels* em um pacote.

Além disso, os comutadores ATM em geral não suportam VCs mutiponto-ponto e

multiponto-multiponto, e por esta razão, a maioria dos comutadores ATM não suportam *VC-merge*.

Segundo [15] ATM-LSRs que não são capazes de realizar *VC-merge* devem utilizar o modo de distribuição *DownStream-On-Demand* e, modo de retenção de *label* conservativo.

2.5.13 Máquina de Estado do LDP

Esta subseção apresenta a máquina de estado LDP definida em [24]. Segundo [24], a especificação do LDP [21] não define uma máquina de estado para o processamento *hop-by-hop* de mensagens LDP. O *internet-draft* considera importante a definição de uma máquina de estado para a interoperabilidade de diferentes implementações do LDP.

Além disso, em [24] é definido uma estrutura de dados que armazena informações sobre os LSPs a serem estabelecidos. Em um LSR, para cada LSP é criado um bloco de controle LSP, com o campos mostrados na tabela 2.1.

ID do *Label Request UpStream* (definido pelo LSR *UpStream*)
ID do *Label Request DownStream* (definido pelo próprio LSR)
Identificador de Sessão do LDP *UpStream*
Identificador de Sessão do LDP *DownStream*
Estado
FEC
Label UpStream (definido pelo LSR em questão)
Label DownStream (definido pelo LSR *DownStream*)
Comportamento se houver um próximo *hop* novo
Identificador de Sessão LDP do novo *Next-Hop DownStream* (usado enquanto se espera pelo *label* do próximo *hop* novo)
Label Request para o próximo *hop* novo (usado enquanto se espera pelo *label* do próximo *hop* novo)

Tabela 2.1: Estrutura de Bloco de Controle LSP

Esta estrutura de dados foi utilizada na especificação do capítulo 4 para se armazenar as informações dos LSPs que fossem estabelecidos durante as simulações. O campo *Estado* da estrutura de controle LSP armazena o estado atual da máquina de estado do LDP, em que o LSP se encontra. Da mesma forma, o estado de um LSP só pode ser observado na especificação do capítulo 4, através dos valores atribuídos ao campo *Estado* do bloco de controle LSP.

A figura 2.12 apresenta o diagrama de máquina de estado LDP definido em [24] para

LSRs ATM, sem capacidade de realizar *VC-merge*. A definição desta máquina leva em consideração a operação do LSR tanto no modo de controle LSR Independente, quanto no modo Ordenado. As ações do LSR, assim como a definição dos eventos da máquina foram baseados nas infomações contidas em [15] e [21].

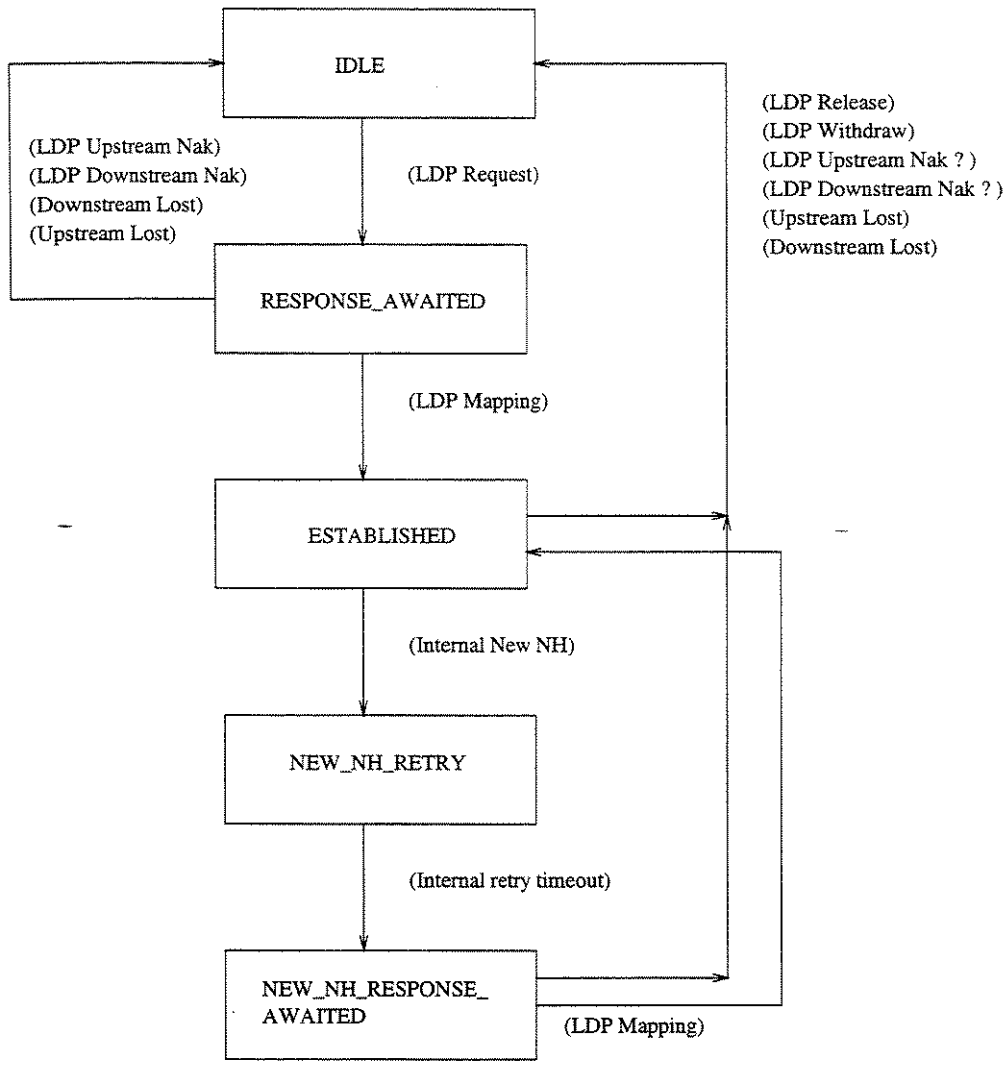

Figura 2.12: Máquina de Estado do LDP para LSRs ATM sem capacidade de *VC-Merge* [24]

Os possíveis estados definidos na figura 2.12 são segundo [24] :

IDLE - este é o estado inicial do LSP quando o registro de bloco de controle

LSP é criado.

RESPONSE-AWAITED - este estado significa que o LSR recebeu uma mensagem de *request* de um LSR *UpStream*, processou o *request*, enviou a mensagem de *request* para o LSR *DownStream* e, está esperando por uma mensagem de *mapping* do LSR *DownStream*.

ESTABLISHED - este estado significa que o LSR recebeu uma mensagem de *mapping* do LSR *DownStream* e que, o LSP está estabelecido e operacional.

NEW-NH-RETRY - este é o estado onde o LSR espera por um temporizador expirar (*retry timer*) e então, tenta estabelecer um LSP através do próximo *hop* novo.

NEW-NH-RESPONSE-AWAITED - este é o estado onde um LSR está no meio do estabelecimento de um novo LSP através do próximo *hop* novo. O LSR enviou uma mensagem de *request* para o próximo *hop* novo e, está esperando que o próximo *hop* novo envie uma mensagem de *mapping*.

Os eventos definidos para a máquina de estado do LDP são :

- *LDP Request* - o LSR recebe uma mensagem de *request* de um LSR *UpStream*.

- *LDP Mapping* - o LSR recebe uma mensagem de *mapping* de um LSR *DownStream*

- *LDP Release* - o LSR recebe uma mensagem de *release* de um LSR *UpStream*.

- *LDP Withdraw* - o LSR recebe uma mensagem de *Withdraw* de um LSR *DownStream*.

- *Internal New NH* - o LSR detecta que há um próximo *hop* novo para uma FEC.

- *LDP UpStream Nak* - o LSR recebe um LDP-Nak de um LSR *UpStream*.

- *LDP DowStream Nak* - o LSR recebe um LDP-Nak de um LSR *DownStream*.

- *Internal Retry Timeout* - o temporizador *retry timer* expirou.

- *UpStream Lost* - o LSR perdeu sua sessão LDP com um LDP *Peer UpStream*

- *DownStream Lost* - o LSR perdeu sua sessão LDP com um *Peer DownStream*.

Capítulo 3

Especificação e Simulação do Classificador de Fluxo do Controlador do IpSwitch

3.1 Introdução

Este capítulo apresenta a especificação formal e a simulação do classificador de fluxo que é parte integrante do controlador de um IpSwitch proposto pelo protocolo IpSwitching [27].

A especificação formal do classificador de fluxo foi realizada em SDL (*Specification and Description Language*) desenvolvida e padronizada pelo ITU-T [16], utilizando-se o SDT (*SDL Design Tool*).

Baseado na especificação do IFMP (*Ipsilon Flow Management Protocol*) [14] e no trabalho desenvolvido em [23] foram criadas quatro versões de especificações que são descritas nas seções 3.2, 3.3, 3.4 e 3.5.

O classificador de fluxo do IpSwitch foi escolhido para se realizar o trabalho de especificação formal porque, como o classificador de fluxo é quem vai decidir por selecionar ou não um fluxo para a comutação, o desempenho do protocolo IpSwitching está diretamente ligado ao tipo de classificador de fluxo utilizado, assim como a sua configuração. Além disso, a escolha do classificador deve ser adequada ao tipo de tráfego no qual ele será usado, para que se obtenha uma porcentagem de comutação o mais próximo possível de 100%, e que ao mesmo tempo, não exija uma quantidade elevada de alocação de VCs do IpSwitch.

A primeira versão de especificação contida na estrutura *Pacote PackIpS1*, figura 3.1 e, descrita na seção 3.2, foi desenvolvida com o objetivo de se criar um analisador de fluxo. O analisador de fluxo tem a função de verificar a incidência de tráfego IP, ou seja, o número de fluxos detectados.

Esta análise significa criar uma tabela de entradas dos possíveis fluxos detectados. Nesta versão, nenhuma análise de classificação de fluxo é realizada, isto é, não se identificam quais são os fluxos que serão selecionados para a comutação. O objetivo desta

abordagem é simplesmente analisar a intensidade do tráfego que se utilizará do classificador, ou seja, o número de fluxos detectados, a média do número de pacotes transmitidos por cada fluxo, assim como a média de pacotes transmitidos por segundo.

A segunda versão contida na estrutura *Pacote PackIpS2*, figura 3.1 e, descrita na seção 3.3, é uma evolução da primeira versão, no sentido de acrescentar mais funções ao analisador de fluxo, *Pacote PackIpS1*, transformando-o em um classificador de fluxo. A segunda versão herda toda a definição do *Pacote PackIpS1* que manipula a tabela de fluxos detectados e, além disso, acrescenta procedimentos para a manipulação de uma nova tabela, a tabela de fluxos comutados.

Na versão *Pacote PackIpS2*, um classificador de fluxo do tipo X/Y, capítulo 2 seção 2.4, é especificado. O objetivo desta evolução é analisar dados estatísticos de desempenho do classificador como o número total de fluxos detectados, o número de fluxos selecionados para a comutação, o número de pacotes transmitidos em nível de camada de rede, o número de pacotes comutados, assim como o desempenho do IpSwitching em relação à variação dos parâmetros X e Y do classificador de fluxo.

A terceira e a quarta versão, as versões contidas em *Pacote PackIpS1_1* e *Pacote PackIpS2_1* da figura 3.2 e, que são descritas nas seções 3.4 e 3.5, são alterações das estruturas de dados das versões 1 e 2, isto é, das estruturas de dados do *Pacote PackIpS1* e do *Pacote PackIpS2*. Estes dois novos pacotes apresentam o mesmo comportamento funcional e estrutura SDL dos pacotes anteriores, como pode ser observado comparando-se as figuras 3.1 e 3.2. A diferença das versões 1 e 2 para as versões 1.1 e 2.1, é que as versões 1.1 e 2.1, utilizam uma estrutura de tabela *hash*, com o objetivo de se otimizar o tempo de busca nas tabelas de fluxos e portanto, o tempo de execução das simulações.

A seção 3.6, MSCs de Simulações da versão 2.1, apresenta os diagramas MSCs (*Message Sequence Chart*)[17] de cinco situações específicas de simulação do classificador de fluxo da versão 2.1. Através destes diagramas, pode-se avaliar o comportamento do classificador diante do recebimento de pacotes, da detecção de um fluxo, da seleção de um fluxo para a comutação, e da detecção de um fluxo inativo.

Na seção 3.7, Resultados das Simulações, são apresentados os dados estatísticos coletados durantes as simulações das especificações das versões 1.1 e 2.1. Através dos dados coletados, pode-se detectar o desempenho do IpSwitching em diferentes configurações do classificador de fluxo do tipo X/Y. Foram usados como entrada do simulador diferentes *traces* de tráfego IP. As versões 1.1 e 2.1 foram escolhidas para as simulações por serem as versões otimizadas em relação aos tempos de simulação.

Finalmente a seção 3.8, apresenta considerações sobre o capítulo 3 e a associação dos resultados obtidos no trabalho com os resultados obtidos em [23].

3.2 Especificação do Analisador de Fluxo (Versão 1)

A proposta de integração IP/ATM do IpSwitching [27] tem como objetivo melhorar o transporte do IP sobre o ATM, tirando proveito das características das duas tecnologias.

Neste trabalho, um enfoque maior é dado para o conceito de fluxo. Conhecer o tipo de

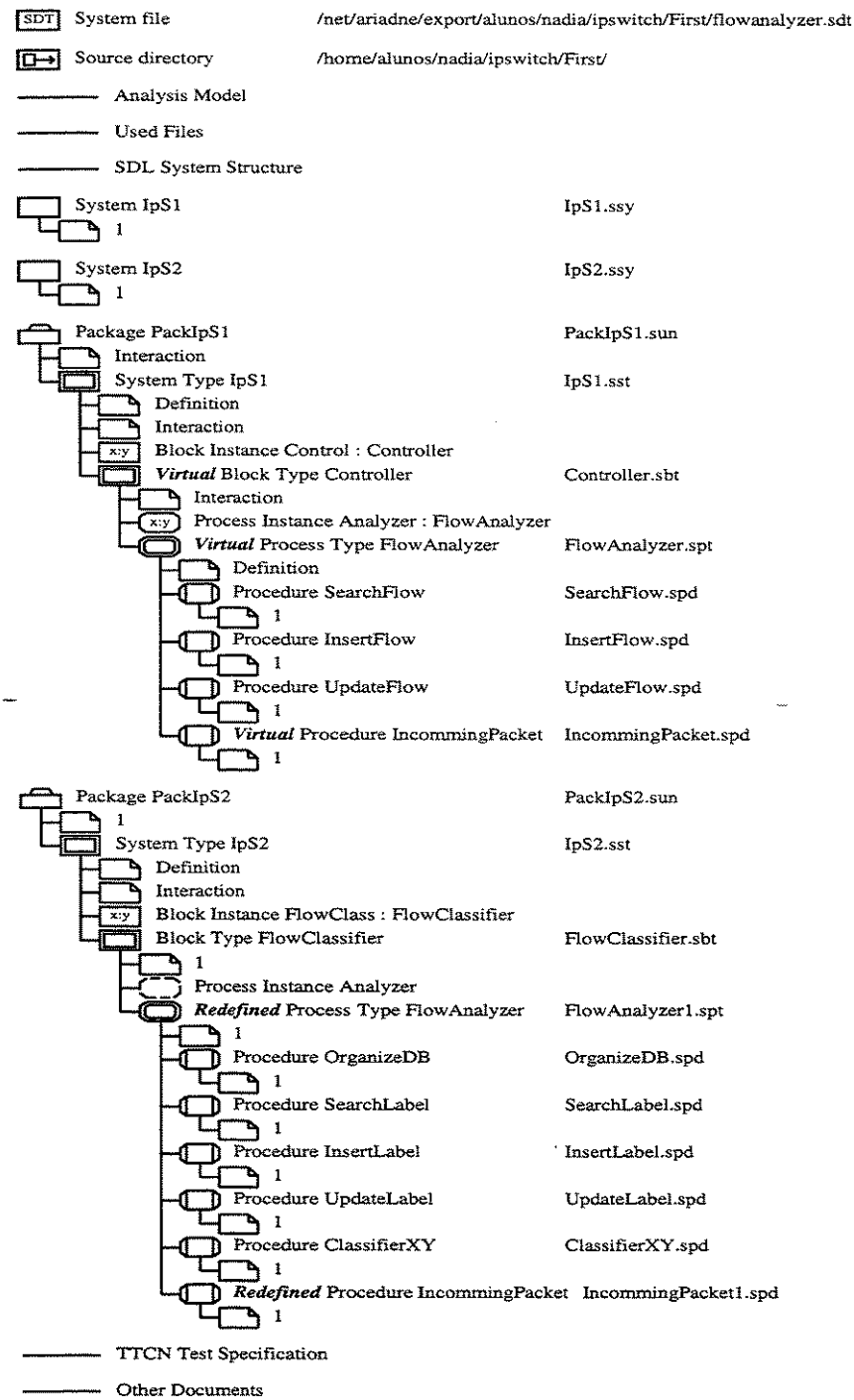

Figura 3.1: Visão do Organizer (SDT) / Versões PackIpS1 e PackIpS2

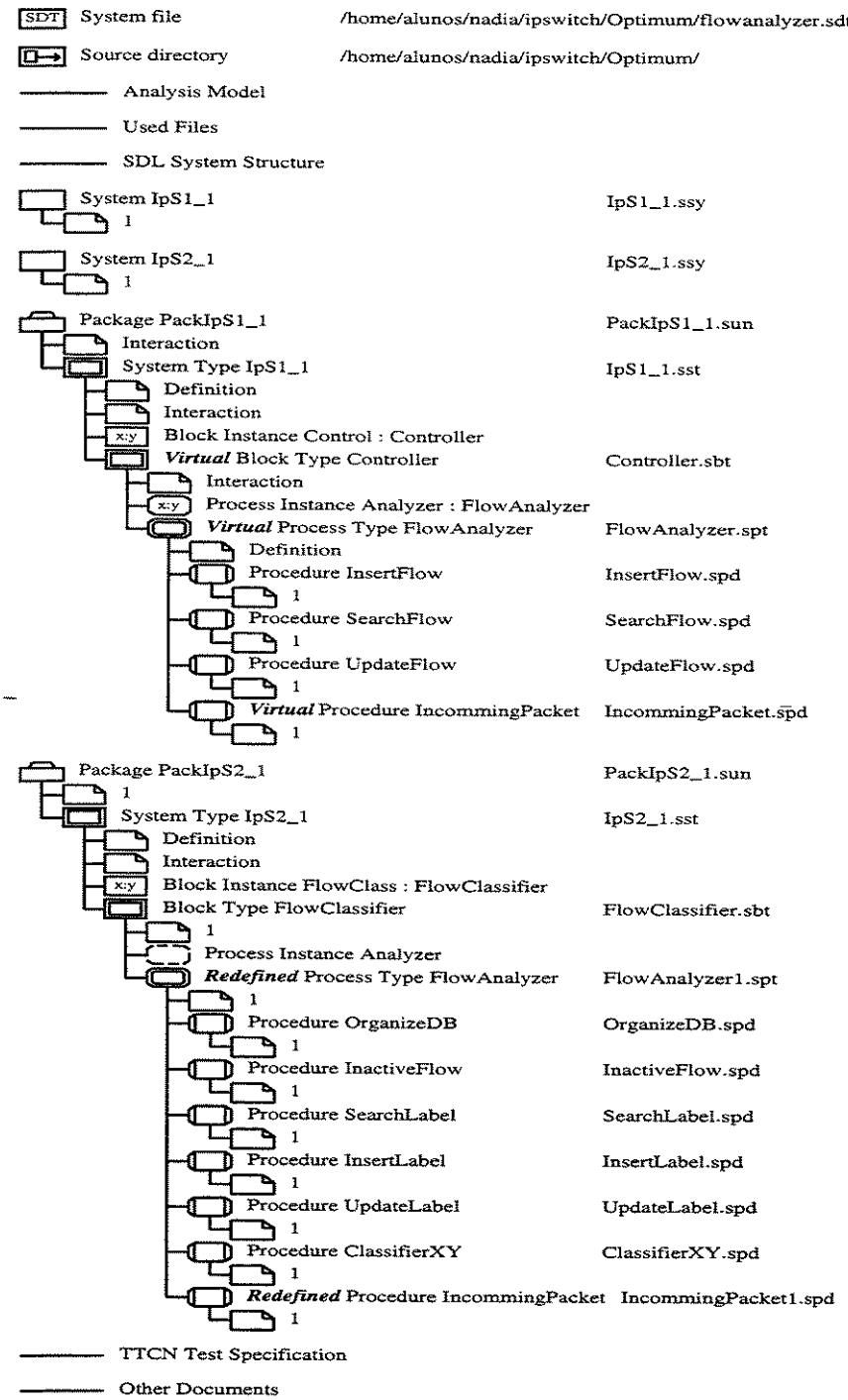

Figura 3.2: Visão do Organizer (SDT) /Versões PackIpS1_1 e PackIpS2_1

tráfego IP que se utilizará do IpSwitching, é importante para a escolha do tipo de classificador de fluxo que será utilizado no controlador do IpSwitch. Tráfegos que apresentam grande incidência de buscas e respostas que transmitam poucos pacotes e entre uma grande quantidade de usuários, terão um número elevado de detecção de fluxos. Já tráfegos IP com pouca quantidade de fluxos de curta duração, e grande quantidade de fluxos que trocam maior quantidade de dados e de longa duração, apresentará um pequeno número de fluxos detectados.

Dados sobre a incidência de tráfego como o número de fluxos detectados e o tempo médio de duração dos fluxos são essenciais para se escolher o classificador de fluxo mais adequado e a configuração dos parâmetros do classificadorO número de fluxos detectados implica na alocação de VCs ATM (*Virtual Channels*), isto é, para cada fluxo selecionado para a comutação, deve haver um VC ATM disponível. Se todos os fluxos de pequena e de longa duração fossem comutados, um grande número de VCs seria necessário. O tempo de duração dos fluxos também deve ser observado pois, alocar VCs para fluxos que transmitirão dados por pouco tempo, não seria conveniente.

Baseada nestes fatos, a primeira versão especifica um analisador de fluxo de operação simples. O analisador de fluxo tem como objetivo analisar a intensidade de tráfego IP em relação ao número de fluxos detectados. Procura-se identificar a quantidade de pacotes transmitidos em média por cada fluxo, assim como a média de pacotes transmitidos por segundo.

Estrutura SDL	Especificação
Package	Pacote PackIpS1
Sistema	*System Type* IpS1
Bloco	*Virtual Block Type* Controller
Processo	*Virtual Process Type* FlowAnalyzer
Procedimentos	*Procedure* SearchFlow, InsertFlow, UpdateFlow e *Virtual Procedure* IncommingPacket

Tabela 3.1: Estruturas SDL da Especificação / Versão 1

Nesta seção a especificação do analisador de fluxo, *Pacote PackIpS1* da figura 3.1, é apresentada com mais detalhes. Esta especificação foi desenvolvida respeitando-se a hierarquia de estruturas do SDL, para que os módulos de operações ficassem organizados de forma a auxiliar em uma possível evolução do sistema.

As estruturas SDL que foram criadas para a especificação do sistema são apresentadas na tabela 3.1 e podem ser observadas no diagrama de hierarquia da figura 3.1. O diagrama

raiz é o pacote *PackIpS1* que contém o sistema *IpS1*. A estrutura *Package* permite que todo um sistema seja herdado em outra especificação.

O bloco *Controller* contido no sistema *IpS1* representa o controlador de um IpSwitch. O processo *FlowAnalyzer* contido no bloco *Controller*, representa o módulo de classificação de fluxo. Nesta especificação, a função de classificação limita-se à analise de tráfego.

A declaração de um sistema, bloco ou processo como um *System Type, Block Type e Process Type*, permite que estas estruturas sejam herdadas por outros sistemas, blocos ou processos. Já a palavra *Virtual*, torna uma estrutura possível de ser redefinida [33][32].

Os procedimentos *SearchFlow*, *InsertFlow*, *UpdateFlow* e *IncommingPacket* são procedimentos pertencentes ao processo *FlowAnalyzer*. Os três primeiros procedimentos são responsáveis pela manutenção dos dados da tabela de entrada de fluxos. Estes três procedimentos manipulam o mesmo conjunto de dados declarados no processo *FlowAnalyzer*. O procedimento *IncommingPacket* especifica o comportamento do IpSwitch diante do recebimento de um pacote IP, realizando chamadas aos três procedimentos de manutenção da tabela de dados.

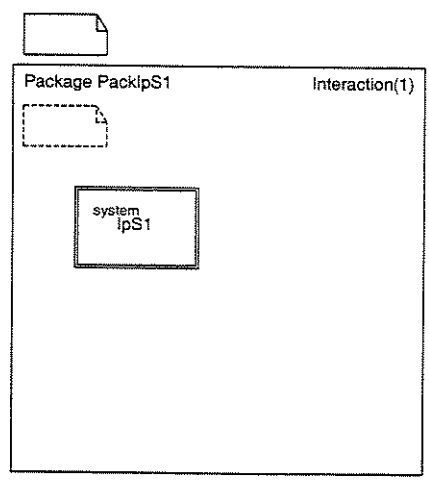

Figura 3.3: Pacote PackIpS1 / Versão 1

Especificou-se o comportamento de apenas um IpSwitch e apenas a função de análise de fluxo. Por este motivo, uma única instância deste sistema foi criada *System IpS1* e, não houve a criação de mais blocos e a possível comunicação entre os mesmos. Não havia a necessidade de se usar todos os níveis de hierarquia da estrutura SDL. O processo *FlowAnalyzer* poderia ter sido declarado diretamente como hierarquia inferior ao sistema *IpS1*, isto é, sem que fosse utilizada a estrutura de bloco.

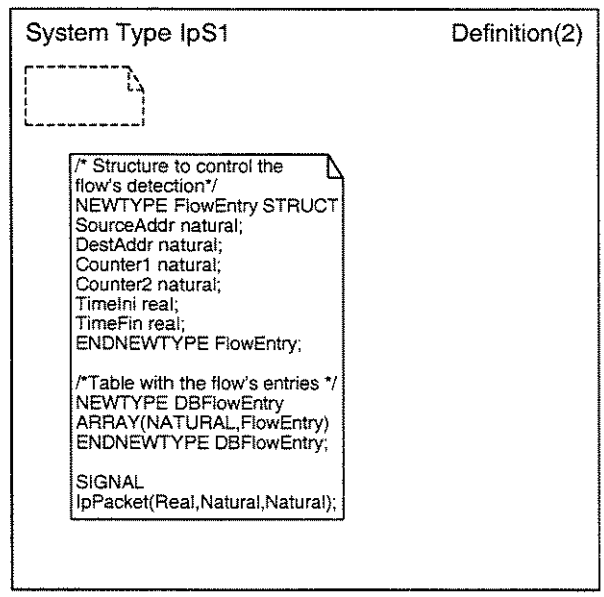

Figura 3.4: Definição do Sistema IpS1 (Pacote IpS1 / Versão 1)

Entretanto, optou-se por utilizar os quatro principais níveis de hierarquia do SDL : sistema, bloco, processo e procedimento com o objetivo de facilitar o entendimento do sistema e para que uma possível evolução neste trabalho, ou em outros trabalhos de pesquisa, pudesse ser realizada mais facilmente.

Com a divisão do sistema em blocos, processos e procedimentos, se um novo bloco precisasse ser acrescentado, nenhuma especificação dos blocos antigos precisaria ser modificada, somente a parte que permitisse a comunicação dos blocos, isto é, a troca de sinais. Um exemplo de evolução e do conceito da orientação a objetos de reutilização, pode ser observado neste trabalho. A especificação do pacote *PackIpS1* é reutilizada na especificação do pacote *PackIpS2*, isto é, o pacote *PackIpS2* herda todas as funções que foram declaradas no pacote *PackIpS1*.

Cada diagrama SDL da especificação e a sua representação gráfica é mostrado e descrito a seguir. A figura 3.3 mostra o *Pacote PackIpS1* e o sistema *IpS1* nele contido.

A figura 3.4 apresenta a definição das estruturas de dados do sistema *IpS1*. Duas estruturas de dados foram criadas no sistema para auxiliar o armazenamento das informações dos fluxos. Considerando-se fluxos do tipo 2, descrito no capítulo 2 na seção 2.4 (figura 2.5), a primeira estrutura criada, *FlowEntry*, possui campos para armazenar o endereço IP origem e o endereço IP destino como identificadores de fluxo.

Um fluxo será considerado como a troca de informações entre dois endereços IP. Considerando dois sistemas finais A e B, o fluxo de dados de A para B, é distinto do fluxo

de dados de B para A. Os outros campos do identificador de fluxo para o tipo de fluxo 2, não são levados em consideração nesta especificação.

A estrutura *FlowEntry* é composta por seis campos : *SourceAddr* que armazena o endereço IP origem, *DestAddr* que armazena o endereço IP destino, *Counter1* e *Counter2* que são contadores auxiliares para o número de pacotes transmitidos no fluxo, *TimeIni* que armazena o instante de transmissão do primeiro pacote do fluxo, e finalmente *TimeFin*, que armazena o instante no qual o último pacote do fluxo foi transmitido.

A segunda estrutura, *DBFlowEntry*, é um vetor (*array*) de entradas sendo estruturas do tipo *FlowEntry* e índice natural. Este vetor funciona como uma tabela de entradas de fluxos do controlador do IpSwitch.

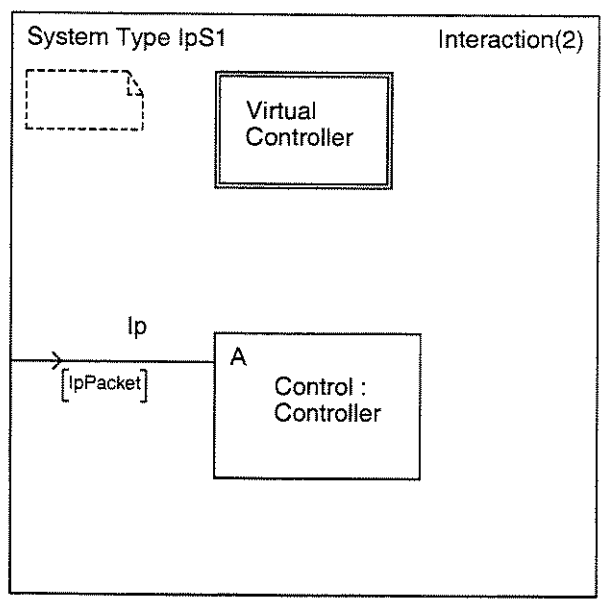

Figura 3.5: Sistema IpS1 (Pacote IpS1 / Versão 1)

Além das duas estruturas, é definido o sinal *IpPacket* com um argumento real e dois argumentos naturais. Este sinal representa um pacote IP, onde o primeiro argumento, do tipo real, é o *TimeStamp* e, o segundo e o terceiro argumento, ambos do tipo natural, são os endereços IP origem e destino do pacote IP.

A figura 3.5 apresenta a especificação do sistema IpS1 que é composto pelo bloco *Virtual Controller* e, por uma instância deste bloco de nome *Control*. Esta instância

recebe dados do ambiente externo através do canal *Ip* onde flui o sinal *IpPacket*.

A figura 3.6 mostra a especificação do bloco *Controller* que contém o processo *Virtual FlowAnalyzer* e, uma instância deste processo de nome *Analyzer*. A instância *Analyzer* comunica-se com o ambiente externo através do *gateway* A por onde flui o sinal *IpPacket*.

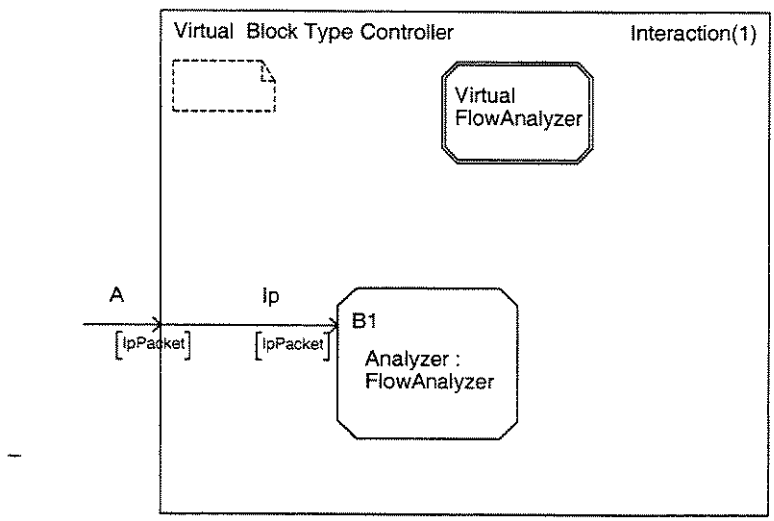

Figura 3.6: Bloco Controller (Sistema IpS1 / Versão 1)

O processo *FlowAnalyzer* mostrado na figura 3.7, é o responsável por organizar os pacotes IP recebidos através do sinal *IpPacket*, em entradas de uma tabela do tipo *DB-FlowEntry*. O conjunto de dados declarados no escopo do processo *FlowAnalyzer* pode ser visualizado por todos os procedimentos nele contido. Em SDL, as variáveis definidas no escopo de um processo, podem ser vistas por todos os seus procedimentos, porém não podem ser vistas por procedimentos pertencentes a outros processos. Esta estrutura funciona de maneira semelhante aos atributos de uma classe que podem ser vistos por todos os seus métodos, mas só pelos seus métodos.

O conjunto de dados declarado é formado por algumas variáveis auxiliares, *TimeStamp, Source, Dest, Resp* e *Aux2*), uma variável de nome *Record* do tipo *DBFlowEntry*, a variável *DbIndex* que armazena o número de fluxos inseridos na tabela *Record* e, a variável *TotPackets* que armazena o número total de pacotes IP transmitido.

As funções do processo *FlowAnalyzer* da figura 3.7 foram divididas e implementadas em procedimentos : *IncommingPacket*, *Search Flow*, *InsertFlow* e, *UpdateFlow* que são

mostrados nas figuras 3.8, 3.9, 3.10 e, 3.11 respectivamente. Esta divisão em procedimentos, tem como objetivo um melhor entendimento do processo devido à sua modularização. Em uma eventual reutilização, somente alguns procedimentos precisariam ser redefinidos e não todo o processo.

De acordo com a figura 3.7, iniciada a operação do processo *FlowAnalyzer*, ele mantém-se no estado *Waiting* até que receba um sinal *IpPacket* do ambiente externo. Cada vez que o sinal for recebido, o procedimento *IncommingPacket* é executado. O estado *Waiting* indica que o controlador do IpSwitch está pronto para receber um pacote IP, isto é, um sinal *IpPacket*.

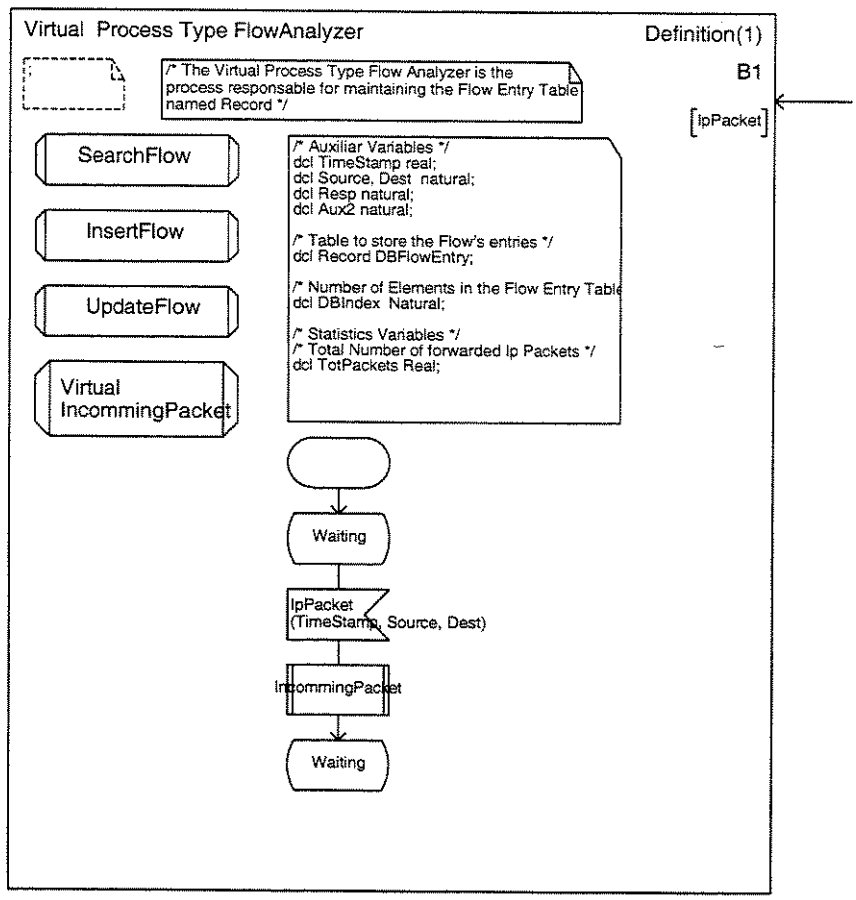

Figura 3.7: Processo FlowAnalyzer (Bloco Controller / Versão 1)

O procedimento *IncommingPacket*, figura 3.8, é responsável pela manipulação da tabela de entrada de fluxos. Ele incrementa a variável *TotPackets* que contabiliza o número total de pacotes IP transmitido e chama o procedimento *SearchFlow*. Se o fluxo já existir

na tabela, os dados do fluxo são atualizados através da chamada do procedimento *UpdateFlow*, caso contrário, um novo fluxo será inserido na tabela *Record* através da chamada do procedimento *InsertFlow*.

Existem duas maneiras de se invocar um procedimento em SDL. A primeira maneira é através da construção *Task* e *Call* quando o procedimento retorna um valor. Um exemplo é a chamada do procedimento *SearchFlow* na figura 3.8. E a segunda maneira através do símbolo *Procedure Call* como a chamada dos procedimentos *UpdateFlow* e *InsertFlow* na figura 3.8.

O procedimento *SearchFlow*, figura 3.9, procura por um fluxo na tabela *Record* correspondente aos endereços IP origem e destino que identificam o fluxo. Se o fluxo for encontrado na tabela, o índice da entrada para este fluxo na tabela é retornado. Caso contrário, o valor 0 é retornado.

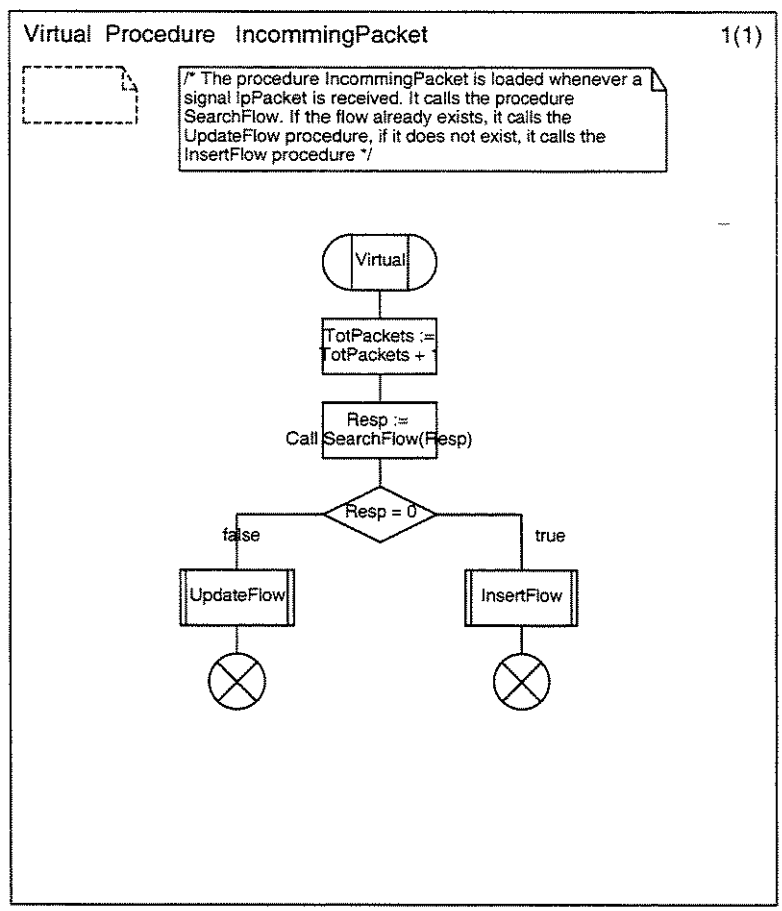

Figura 3.8: Procedimento IncommingPacket (Processo FlowAnalyzer / Versão 1)

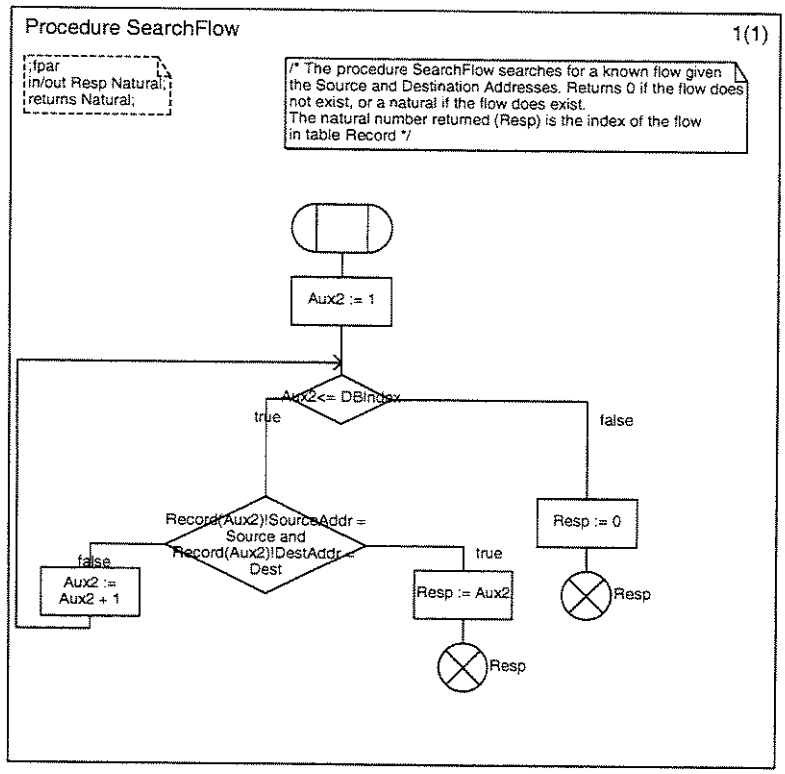

Figura 3.9: Procedimento SearchFlow (Processo FlowAnalyzer / Versão 1)

O procedimento *InsertFlow*, figura 3.10, adiciona um novo fluxo na tabela *Record*, incrementando o número de elementos da tabela *DBIndex* e, inicializando os campos da estrutura de entrada de fluxo com os valores dos endereços origem e destino (*Source* e *Dest*) recebidos no sinal *IpPacket*. Os contadores auxiliares *Counter1* e *Counter2* são inicializados com o valor 1, indicando a chegada do primeiro pacote do fluxo, e o tempo inicial e final com o valor do instante de chegada do pacote. A inserção de um novo fluxo se dá sempre após o elemento que esta na última posição da tabela.

O procedimento *UpdateFlow*, figura 3.11, atualiza o valor dos contadores auxiliares *Counter1* e *Counter2* incrementando-os em uma unidade para indicar que um pacote deste fluxo foi transmitido. O campo *TimeFin*, recebe o valor do *TimeStamp* de chegada deste novo pacote.

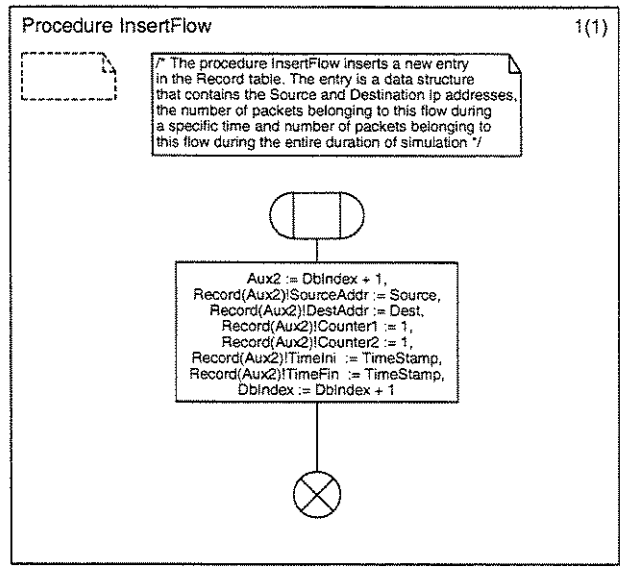

Figura 3.10: Procedimento InsertFlow (Processo FlowAnalyzer / Versão 1)

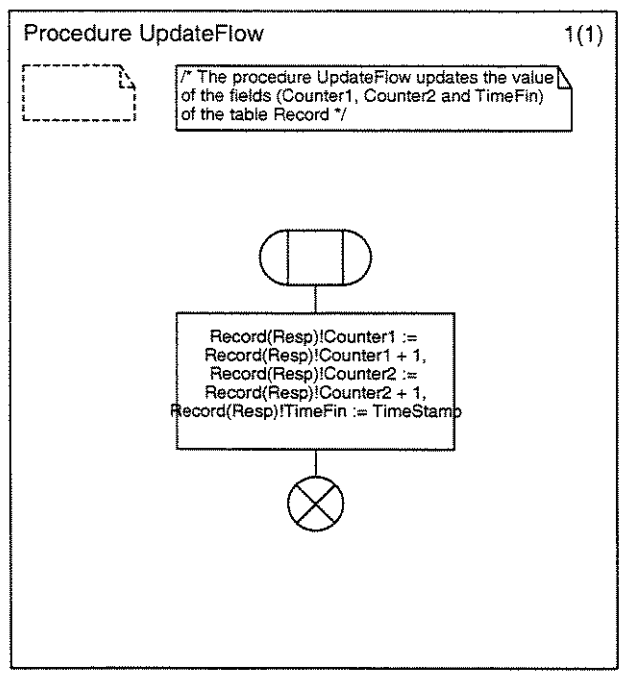

Figura 3.11: Procedimento UpdateFlow (Processo FlowAnalyzer / Versão 1)

3.3 Especificação do Classificador de Fluxo X/Y (Versão 2)

O objetivo principal do IpSwitching, capítulo 2 seção 2.4, é comutar o maior número de pacotes IP diretamente no hardware de camada 2, e transmitir o menor número possível de pacotes IP em nível de camada 3. Para que este objetivo seja alcançado, a especificação do classificador de fluxo é essencial.

O classificador de fluxo é que vai decidir se um fluxo deve ou não ser comutado. Esta seção apresenta a especificação do classificador de fluxo X/Y do controlador do IpSwitch. Procura-se analisar além da quantidade de fluxos detectados, o número de fluxos que, através do critério de seleção do classificador X/Y, serão selecionados para comutação. Além disso, deseja-se obter dados estatísticos sobre o número total de pacotes transmitidos, o número total de pacotes comutados e número de fluxos que tornaram-se inativos.

Esta nova versão de especificação parte da versão 1, isto é, ela herda toda a estrutura SDL declarada no pacote *PackIpS1*, figura 3.1, que realiza a detecção de fluxos de um tráfego IP. Porém acrescenta-se novas estruturas de dados e funções para possibilitar a manipulação de uma nova tabela, a tabela de fluxos comutados. As funções do classificador de fluxo também são especificadas.

De acordo com a figura 3.1, pode-se visualizar a hierarquia do novo pacote criado, o pacote *PackIpS2*. A tabela 3.2 apresenta as estruturas SDL da especificação da versão 2.

Estrutura SDL	Especificação
Package	PackIpS2
Sistema	*System Type* IpS2
Bloco	*Block Type* FlowClassifier
Processo	*Redefined Process Type* FlowAnalyzer
Procedimentos	*Procedure* OrganizeDB, SearchLabel, InsertLabel, UpdateLabel, ClassifierXY e *Redefined Procedure* IncommingPacket

Tabela 3.2: Estruturas SDL da Especificação/ Versão 2

O pacote *PackIpS2* que contém a especificação de um novo sistema chamado de *IpS2* pode herdar qualquer uma das estruturas SDL definidas no pacote *PackIpS1*, através do comando *USE PackIpS1* mostrado na figura 3.12.

A função de cada nova estrutura SDL, assim como as novas estruturas de dados e re-

definições de processos e procedimentos, são apresentadas através dos diagramas a seguir.

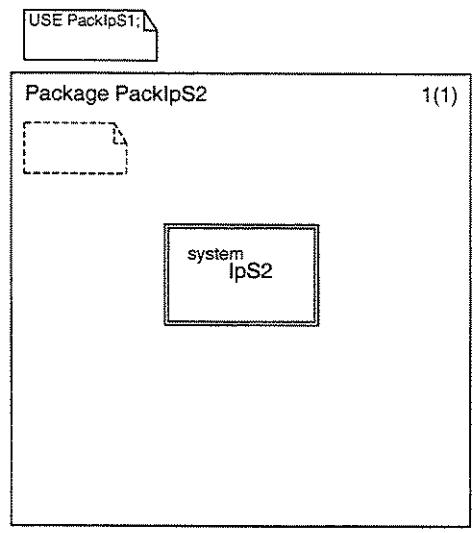

Figura 3.12: Pacote PackIpS2/Versão 2

O pacote *PackIpS2* cria um novo sistema, o sistema *IpS2*. O sistema *IpS2* herda as estruturas de dados declaradas no sistema *IpS1*, através do comando *inherits* como mostrado na figura 3.13. Portanto, as estruturas *FlowEntry*, *DBFlowEntry* e o sinal *IpPacket* podem ser usados no sistema *IpS2*. Além dessas estruturas, o sistema *IpS2* adiciona duas novas estruturas, *Label* e *DBLabel*.

A estrutura de nome *Label* é uma estrutura que tem a função de armazenar os dados referentes aos fluxos selecionados para a comutação. Para isso, utiliza 6 campos : *FlowIdentifier1*, *FlowIdentifier2*, *Counter1*, *Counter2*, *TimeIni* e *TimeFin* da mesma forma que a estrutura *FlowEntry* da versão 1 que armazena os dados sobre os fluxos detectados. Os endereços origem e destino que identificam o fluxo são armazenados em *FlowIdentifier1* e *FlowIdentifier2*. Os campos *Counter1* e *Counter2* são variáveis auxiliares para a contabilização do número de pacotes transmitidos no fluxo após ter sido selecionado para comutação direta em nível de camada 2. E finalmente, os campos *TimeIni* e *TimeFin* armazenam o instante em que o primeiro pacote do fluxo foi comutado, e o instante que registra a última comutação de pacote do fluxo respectivamente.

Além das estruturas de dados, o sistema *IpS2* também herda todas as estruturas SDL do sistema *IpS1* da figura 3.5 e, cria um novo bloco de nome *FlowClassifier*, com uma instância *FlowClass* que se comunica com o ambiente externo através do canal Ip2 onde

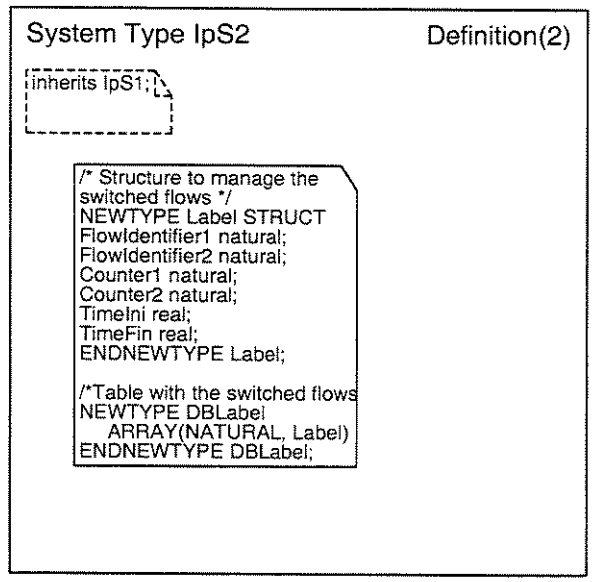

Figura 3.13: Definição do Sistema IpS2 (Pacote IpS2/Versão 2)

flui o sinal IpPacket, como mostrado na figura 3.14. O sinal *IpPacket* é o mesmo sinal declarado na versão 1 e que foi herdado, porém agora podendo trafegar pelo canal *IpS2* além do canal *Ip* declarado no sistema *IpS1*.

O bloco *FlowClassifier* da figura 3.15 foi criado com o objetivo de se representar o controlador de um IpSwitch e seu classificador de fluxo. Ele herda as estruturas declaradas no bloco *Controller* da versão 1, figura 3.6. O bloco *Controller* é herdado pois ele realiza a detecção de fluxo que é necessária para a especificação de um classificador de fluxo. Dos fluxos detectados, através dos parâmetros de seleção do classificador de fluxo X/Y, os fluxos serão ou não selecionados para a comutação de seus pacotes.

Na figura 3.16, o processo *FlowAnalyzer* é redefinido como o intuito de se acrescentar as funções de classificação de fluxo ao analisador de fluxo da versão 1. O gateway A e a instância *Analyzer* aparecem em linhas pontilhadas na figura 3.15, para indicar que são estruturas pré-definidas anteriormente e que serão modificadas.

O processo *FlowAnalyzer*, figura 3.16, redefine o procedimento *IncommingPacket* em relação à figura 3.7 e, adiciona 5 novos procedimentos : *OrganizeDB*, *SearchLabel*, *InsertLabel*, *UpdateLabel* e *ClassifierXY* das figuras 3.17, 3.18, 3.19, 3.20 e, 3.22 respectivamente.

Na figura 3.17 o procedimento *OrganizeDB* tem a função de realizar uma verificação periódica tanto da tabela de fluxos detectados, como da tabela de fluxos comutados.

Os procedimentos *SearchLabel*, *InsertLabel* e *UpdateLabel* são responsáveis pela manu-

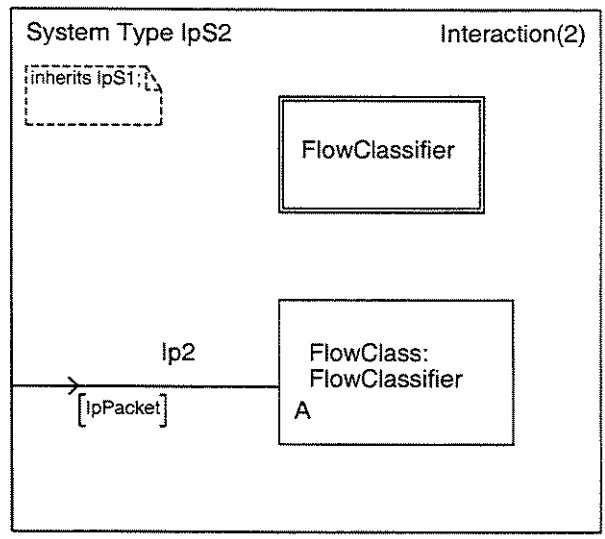

Figura 3.14: Sistema IpS2 (Pacote PackIpS2 / Versão 2)

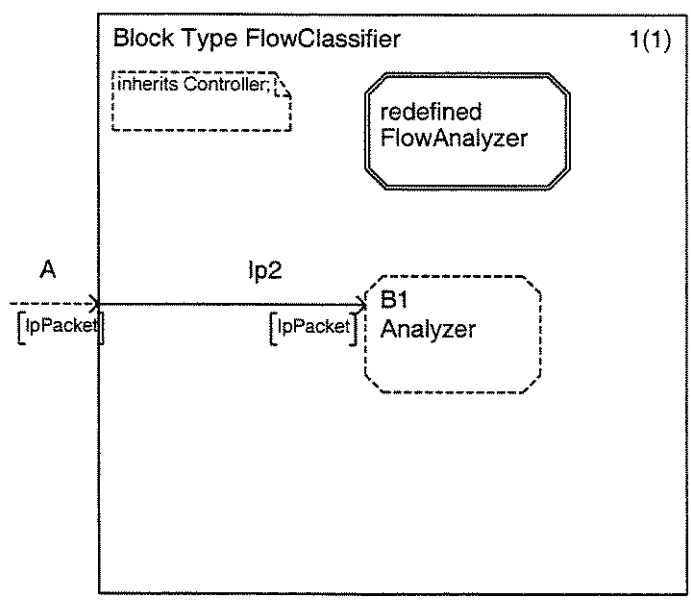

Figura 3.15: Bloco FlowClassifier (Sistema IpS2 / Versão 2)

tenção da tabela de fluxos comutados. O procedimento *ClassifierXY* define os parâmetros de classificação de fluxo para um classificador do tipo X/Y.

Os procedimentos declarados no processo *FlowAnalyzer* da versão 1 : *SearchFlow*, *InsertFlow*, *UpdateFlow*, figura 3.7, continuam sendo utilizados para a manutenção da tabela de fluxos detectados, *Record*.

Na redefinição do processo *FlowAnalyzer* novas variáveis foram declaradas observando-se que as variáveis declaradas na versão 1, continuam sendo utilizadas, porém a maioria é manipulada pelos procedimentos também declarados na versão 1. Uma tabela de nome *Label* do tipo *DBLabel* como mostrado na figura 3.16, foi declarada para armazenar os dados sobre os fluxos comutados. A variável *DBIndex2* é um índice para a tabela *Label* e armazena o número de elementos inseridos nesta tabela. Algumas novas variáveis auxiliares também foram acrescentadas ao processo : *Aux*, *Aux3*, *Ident1*, *Ident2*, *Result* e, *TotSwitched* que armazena dados estatísticos sobre o número total de pacotes comutados.

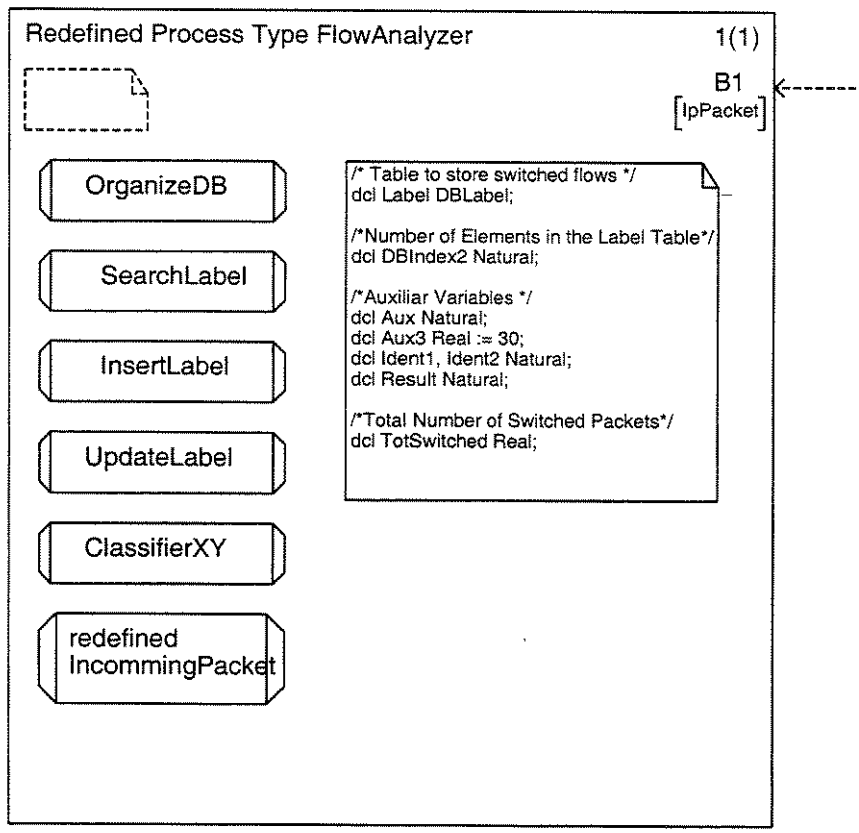

Figura 3.16: Processo FlowAnalyzer Redefinido (Bloco FlowClass / Versão 2)

A variável auxiliar *Aux3*, é responsável por indicar ao procedimento *OrganizerDB*

a periodicidade em que as tabelas dos fluxos devem ser verificadas. Desta forma, em uma simulação, se a variável *Aux3* for declarada com o valor inicial de 30 segundos por exemplo, isto significa que esta operação de manutenção das tabelas será realizada de 30 em 30 segundos.

O procedimento *OrganizerDB* mostrado na figura 3.17, é o procedimento responsável por realizar a verificação de todos os fluxos, tanto dos fluxos candidatos como dos fluxos comutados.

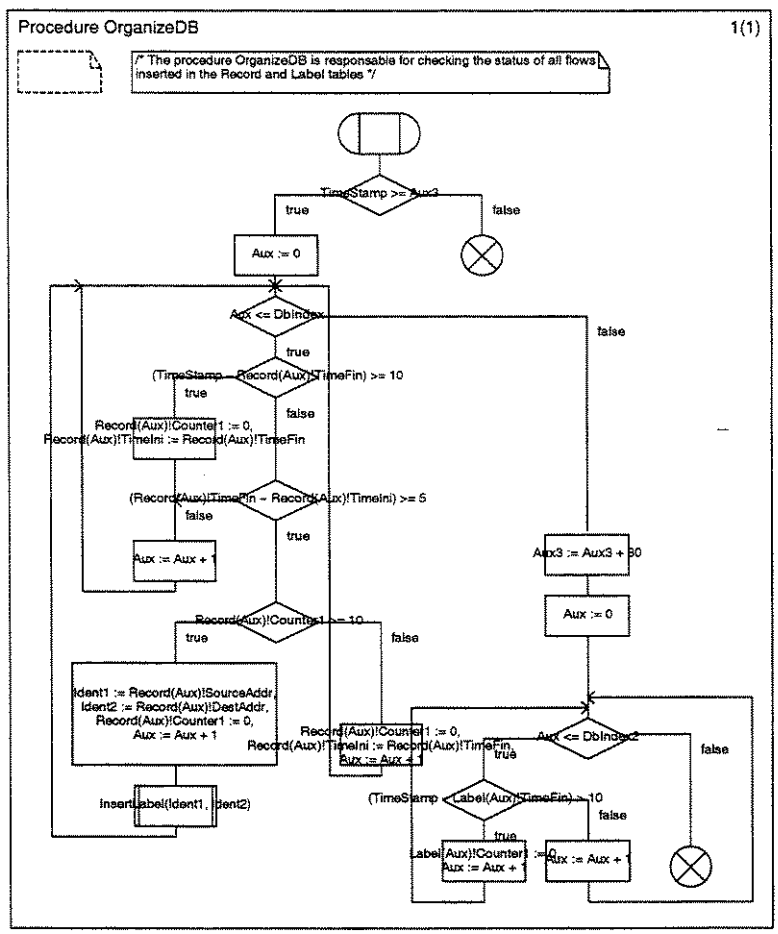

Figura 3.17: Procedimento OrganizeDB (Processo FlowAnalyzer / Versão 2)

A função do procedimento *OrganizerDB* sobre a tabela *Record* da versão 1, ocorre da seguinte forma : dado um valor Z igual a *Aux3* segundos, o procedimento *OrganizeDB* verifica de Z em Z segundos, se há fluxos na tabela de fluxos candidatos que, foram inseridos há mais de Y segundos e que, ainda não apresentaram X pacotes. Estes fluxos

são reinicializados e o comportamento desses fluxos é analisado por um novo intervalo de tempo Z. Se por exemplo os valores X, Y e Z forem 10, 5 e 20 segundos respectivamente, assume-se que, se passado 20 segundos este fluxo não conseguiu atingir o valor de 10 pacotes em 5 segundos, este fluxo não é um bom fluxo para comutação.

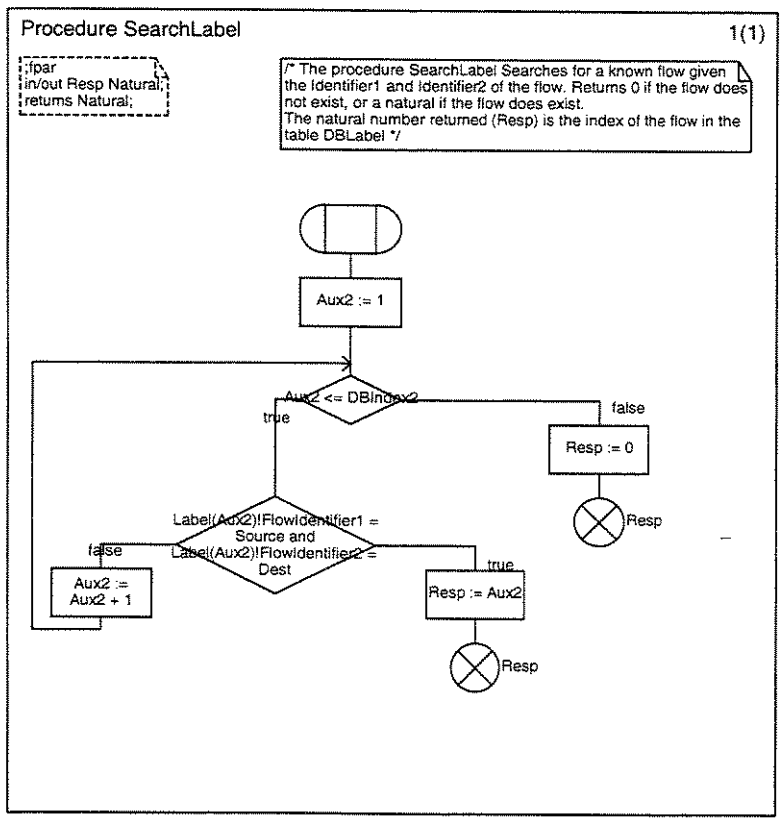

Figura 3.18: Procedimento SearchLabel (Processo FlowAnalyzer / Versão 2)

A inicialização dos fluxos é utilizada como sinônimo de descarte em tabela. Quando um fluxo é descartado, alguns instantes depois ele pode voltar a transmitir dados. Optou-se por apenas reinicializar os campos que armazenam o número de pacotes, o tempo do recebimento do primeiro e do último pacote, ao invés de remover o fluxo fisicamente da tabela. Este procedimento, faz com que menos acessos às tabelas sejam realizados, entretanto, mantém-se a tabela de fluxos candidatos com um número grande de entradas. O tempo de simulação foi menor quando esta decisão foi utilizada, pois cada remoção física, implicaria em uma reorganização de toda a tabela.

O mesmo fluxo pode vir a apresentar um tráfego mais intenso posteriormente, portanto, descartam-se os dados X e Y adquiridos até o momento em questão e, novos dados

começam a ser coletados como se o fluxo tivesse sido detectado naquele instante. Passados mais Z segundos, o mesmo fluxo, agora com outros valores para o número de pacotes e tempo em segundos, é analisado novamente.

Já a operação do procedimento *OrganizeDB* sobre a tabela de fluxos comutados, tem a função de detectar se alguns dos fluxos que já foram selecionados como bons para a comutação, não se tornaram inativos. Um valor W é determinado de forma que, se a diferença entre o tempo da simulação e o tempo da comutação do último pacote do fluxo for maior do que W, o fluxo é considerado inativo e é então descartado. O descarte de uma entrada da tabela *DBLabel* também é realizado reinicializando-se valores de alguns de seus campos para buscar uma otimização do tempo de simulação.

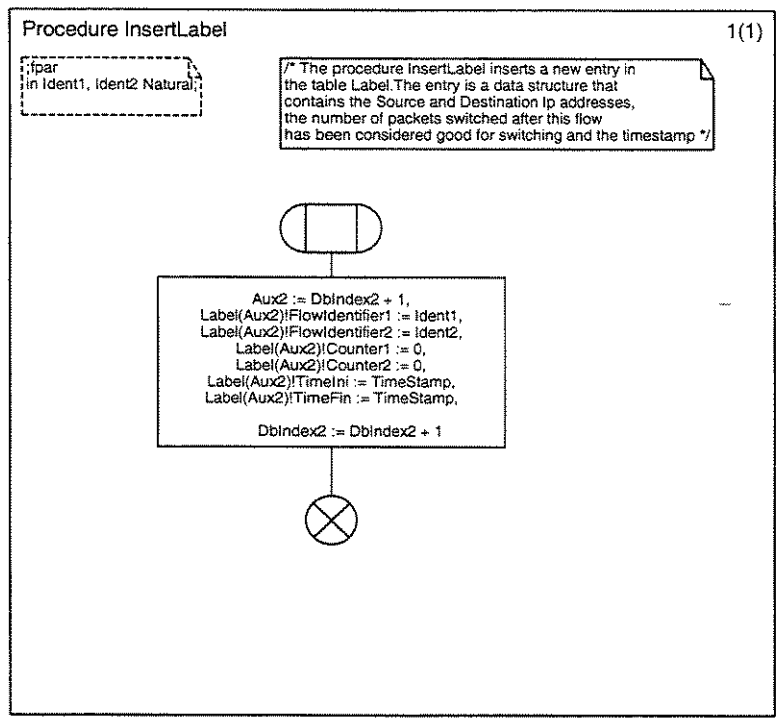

Figura 3.19: Procedimento InsertLabel (Processo FlowAnalyzer / Versão 2)

Os procedimentos *SearchLabel*, *InsertLabel* e *UpdateLabel* das figuras 3.18, 3.19 e 3.20, são responsáveis pela consulta, inserção e atualização de entradas na tabela de fluxos comutados. Eles apresentam a mesma estrutura e operação dos procedimentos *SearchFlow*, *InsertFlow* e *UpdateFlow* da versão 1 figuras 3.9, 3.10 e 3.11 respectivamente, entretanto, as suas operações são realizadas sobre a tabela *Label* e não sobre a tabela *Record*.

O procedimento *SearchLabel*, figura 3.18, procura por um fluxo na tabela *Label* dados os endereços IP origem e destino que identificam o fluxo. Se o fluxo for encontrado na

tabela *Label*, o índice da entrada deste fluxo na tabela é retornado. Caso contrário, o valor 0 é retornado.

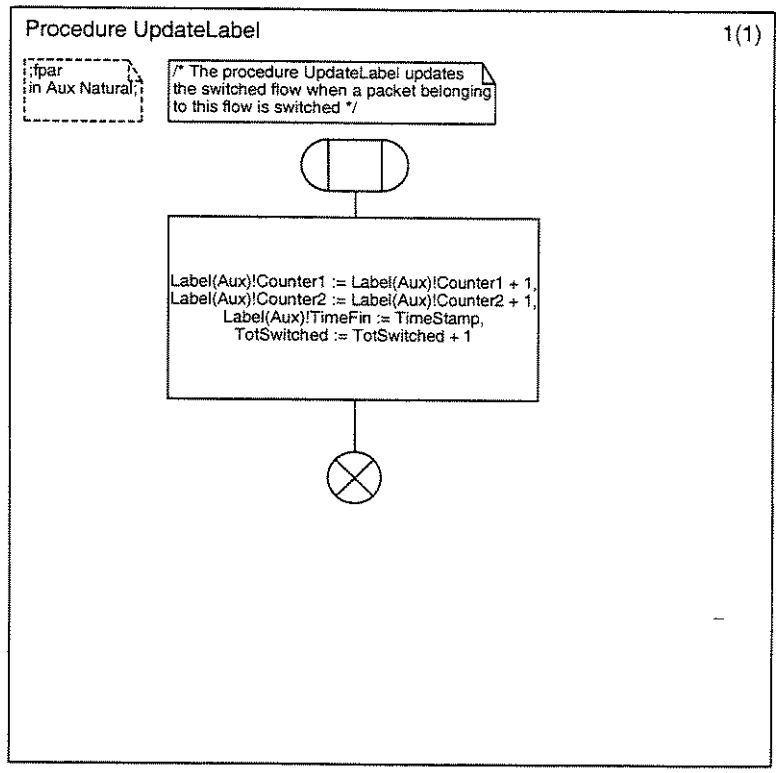

Figura 3.20: Procedimento UpdateLabel (Processo FlowAnalyzer / Versão 2)

O procedimento *InsertLabel*, figura 3.19, adiciona um novo fluxo na tabela *Label* incrementando o número de elementos da tabela *DBIndex2* e, incializando os campos da estrutura de entrada de fluxo, *FlowIdentifier1* e *FlowIdentifier2* com os valores dos endereços origem e destino. Os contadores auxiliares *Counter1* e *Counter2* são inicializados com o valor 0, isto é, nenhum pacote para este fluxo foi comutado até o momento. Assim como a mensagem de redirecionamento do IFMP (*Ipsilon Flow Management Protocol*) não espera reconhecimento, o recebimento do primeiro pacote no fluxo comutado, indica que a operação se realizou com sucesso e que o fluxo começou a ser comutado no hardware de camada 2. A inserção de um novo fluxo comutado se dá sempre após a última posição da tabela.

O procedimento *UpdateLabel*, figura 3.20, atualiza o valor dos contadores auxiliares *Counter1* e *Counter2* incrementando-os em uma unidade para indicar que um pacote deste fluxo foi comutado. O campo *TimeFin* recebe o valor do *TimeStamp* deste novo pacote.

O procedimento *ClassifierXY*, figura 3.22, estabelece os valores dos parâmetros X e

Y, e verifica se uma dada entrada da tabela de fluxos candidatos atingiu esses valores. O índice do fluxo na tabela *Record* é passado como parâmetro de entrada, sendo o valor de retorno igual a 1 se o fluxo for selecionado para comutação, e 2 se o fluxo não atendeu aos requisitos estabelecidos no classificador X/Y.

O procedimento *IncommingPacket* mostrado na figura 3.21, foi redefinido para administrar o processo de seleção de fluxos. Estre procedimento na versão 1, figura 3.8, manipulava apenas as chamadas aos procedimentos *SearchFlow*, *InsertFlow* e *UpdateFlow* que realizavam a análise do tráfego em relação a fluxos gerados. Agora na versão 2, ele também manipula a chamada aos novos procedimentos declarados, o *OrganizeDB*, *SearchLabel*, *InsertLabel*, *UpdateLabel* e *ClassifierXY*. O procedimento *IncommingPacket* continua sendo ativado cada vez que um sinal *IpPacket* chega ao sistema, isto é, a cada pacote que chega em um IpSwitch.

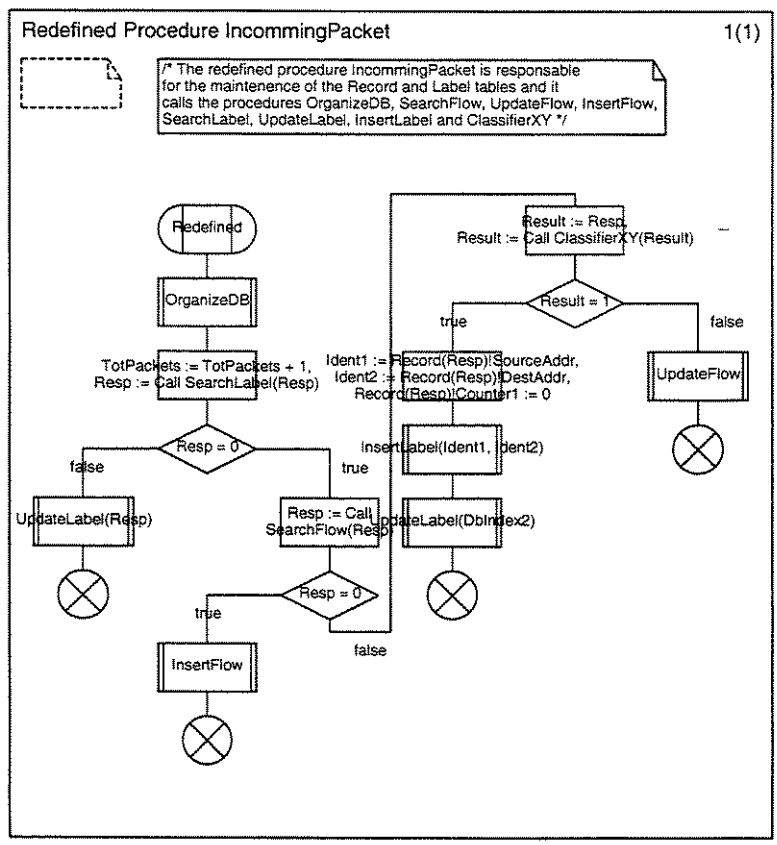

Figura 3.21: Procedimento IncommingPacket Redefinido (Processo FlowAnalyzer / Versão 2)

De acordo com a figura 3.21, recebido um sinal *IpPacket*, o procedimento *OrganizeDB*

é chamado para a verificação dos fluxos das tabelas *Record* e *Label*. Através da chamada do procedimento *SearchLabel* da figura 3.18, os dados do pacote recebido (endereços IP origem e destino que identificam um fluxo), são procurados na tabela de fluxos comutados, tabela *Label*. Caso este fluxo já tenha sido selecionado para a comutação, os seus dados são atualizados com a chamada do procedimento *UpdateLabel2*.

Caso o fluxo analisado não estiver sido selecionado para a comutação, procura-se pelo identificador deste fluxo na tabela de fluxos candidatos, tabela *Record*, chamando-se o procedimento *SearchFlow* da figura 3.9. Se este mesmo fluxo não for um fluxo candidato, isto é, não existe na tabela *Record*, ele será inserido como um novo fluxo candidato, através do procedimento *InsertFlow* da figura 3.10.

Com a chamada do procedimento *ClassifierXY* da figura 3.22, se o fluxo já existir na tabela de fluxos candidatos, é verificado se os parâmetros X e Y foram alcançados por este fluxo, para que possa ser considerado um fluxo bom para comutação.

Desta forma, se o fluxo atingiu os parâmetros, ele é inserido na tabela de fluxos selecionados pelo procedimento *InsertLabel* da figura 3.19. Se ainda não atingiu os parâmetros X pacotes em Y segundos, seus dados são atualizados na tabela de fluxos candidatos pelo procedimento *UpdateFlow* da figura 3.11.

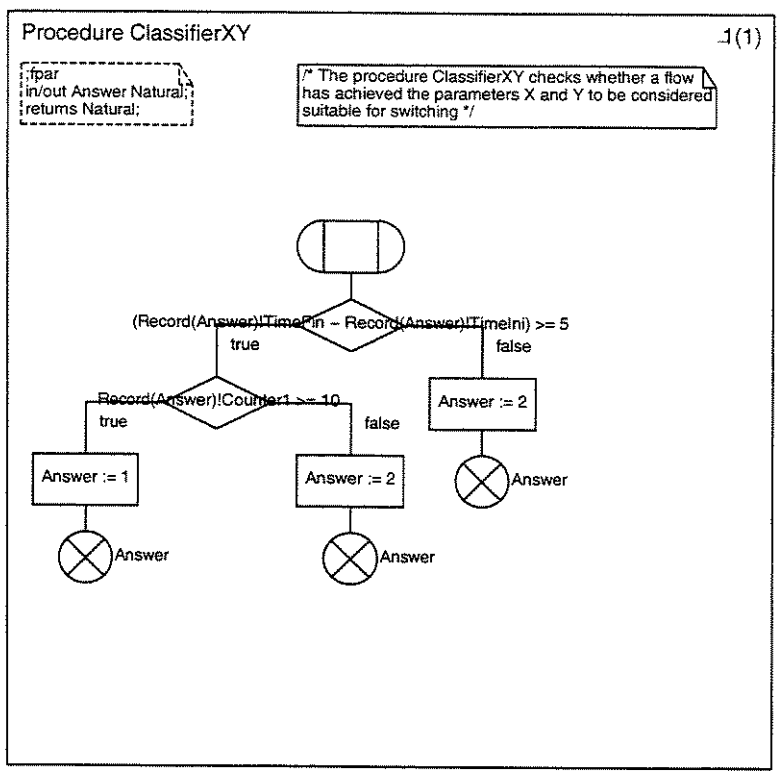

Figura 3.22: Procedimento ClassifierXY (Processo FlowAnalyzer / Versão 2)

3.4 Especificação do Analisador de Fluxo (Versão 1.1)

Na especificação da versão 1 descrita na seção 3.1, o tempo de simulação para arquivos de trace de tráfego IP pequenos foram significativos. Para a simulação tendo como entrada um arquivo de tráfego IP de 5 minutos, o simulador levou 10 horas aproximadamente para obter os resultados finais. Esperava-se tempos de simulação razoáveis pois, conforme o trace vai alimentando o simulador, a tendência é que o número de fluxos candidatos e o número de fluxos comutados aumentem.

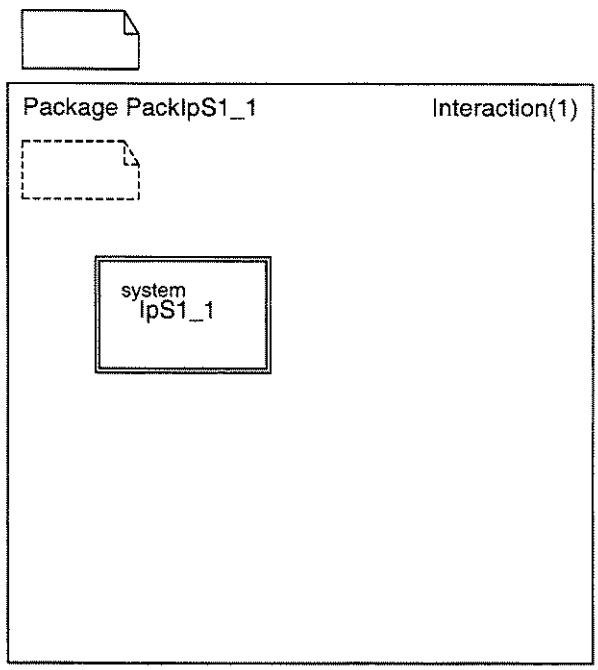

Figura 3.23: Pacote PackIpS1_1 / Versão 1.1

As buscas nas tabelas foram implementadas de forma sequencial, isto é, para se encontrar um fluxo dado seu identificador, o simulador começa uma procura sequencial desde o início da tabela, até que se encontre o fluxo procurado ou, se atinja o final da tabela. Como os fluxos novos são inseridos sempre no final da tabela, e logo que detectados apresentam um provável tráfego, estas buscas se tornam exaustivas.

Com o objetivo de minimizar o processamento nas buscas a tabelas grandes, o sistema da versão 1 teve as suas estruturas de dados remodeladas. Ao invés de se utilizar uma tabela de acesso sequencial, uma tabela hash foi utilizada. O tempo de simulação

utilizando a versão otimizada descrita nesta seção, foi reduzido em mais de 50%. Para *traces* de 5 minutos, o tempo de simulação foi reduzido de aproximadamente 10 horas, para aproximadamente 4 horas de simulação.

É importante salientar que, nem o comportamento funcional nem a estrutura SDL da versão 1, foram modificadas como mostrado na figura 3.2, somente as suas estruturas de dados . Comparando-se a figura 3.1 com a figura 3.2, pode-se verificar que a estrutura SDL para o pacote *PackIpS1* continua exatamente a mesma. Somente o pacote *PackIpS1* foi renomeado para *PackIpS1_1* e, o sistema *IpS1* para *IpS1_1* com o objetivo de se diferenciar uma versão de especificação da outra.

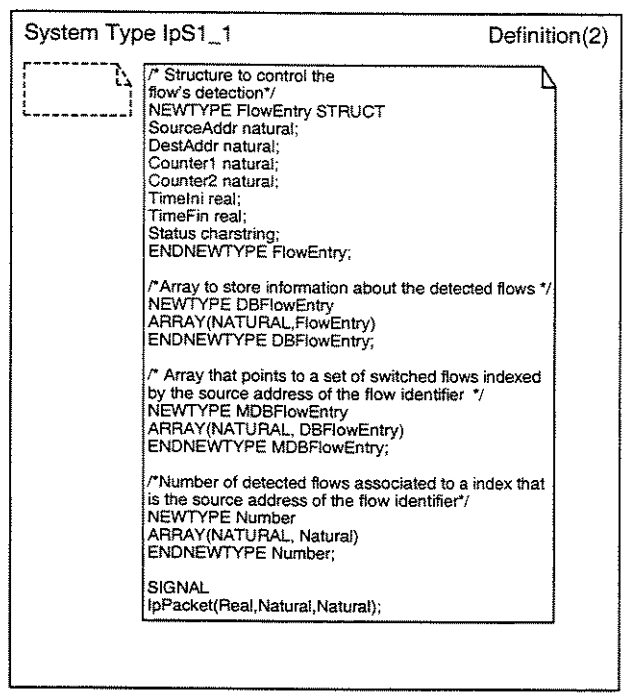

Figura 3.24: Definição do Sistema IpS1_1 (Pacote PackIpS1_1 / Versão 1.1)

Os procedimentos que manipulavam os dados da tabela de entrada de fluxos, foram alterados para se possibilitar o acesso aos dados armazenados na nova estrutura de dados.

Esta seção descreve as alterações realizadas nas estruturas de dados e estrutura SDL da versão 1 do anlisador de fluxo, seção 3.1. Os blocos, processos e procedimentos que não forem apresentados nesta seção, nao sofreram modificações. A versão 1.1 não é herdada da versão 1, mas sim uma modificação.

O pacote *PackIpS1* foi renomeado para pacote *PackIpS1_1*, figura 3.23. O sistema *IpS1*

IP Origem (Primeiro Índice)	IP Destino (Em Ordem Crescente)
1	1, 2, 4, 6 e 8
2	1, 3, 5 e 8
3	3, 4 e 9
...	...

Tabela 3.3: Organização dos Fluxos em uma Tabela Hash

Sigla	Status	Descrição do Status
A	Active	Fluxo Ativo
C	Created	Fluxo Recém Criado
S	Switched	Fluxo Selecionado para Comutação
N	New Period	Fluxo Reinicializado

Tabela 3.4: Descrição dos Status de Fluxos

também foi renomeado para sistema *IpS1_1* e sofreu alterações na estrutura de dados que armazena informações sobre os fluxos detectados como mostrado na figura 3.24.

A estrutura de tabela hash *MDBFlowEntry* criada no sistema *IpS1_1*, figura 3.24, continua tendo a mesma função da estrutura de tabela *DBFlowEntry* da versão 1, figura 3.4. Ela armazena as mesmas informações, porém o endereço IP de origem serve como índice que aponta para um vetor de estruturas *FlowEntry* que por sua vez, armazena os fluxos em ordem crescente de acordo com o seu endereço IP destino. Desta forma, todos os fluxos relacionados ao endereço IP origem 1, por exemplo, teriam como primeiro índice o número 1. Uma busca ao fluxo (1,2) por exemplo, seria sequencial e realizada somente entre os fluxos que tivessem o índice 1 e não em toda a tabela. A tabela 3.3 apresenta a organização dos dados em uma tabela hash.

Na estrutura *FlowEntry*, o campo *Status* foi acrescentado com o objetivo de se armazenar o status de um fluxo na tabela de fluxos candidatos. A tabela 3.4 mostra os valores que podem ser atribuídos ao campo status.

A nova estrutura criada de nome *MDBFlowEntry*, figura 3.24, é um vetor de estruturas do tipo *DBFlowEntry*, que por sua vez é um vetor de estruturas *FlowEntry*. O endereço IP origem é um índice do tipo natural para o vetor *MDBFLowEntry* e, o endereço IP destino serve de índice para a inserção ordenada no vetor *DBFlowEntry*.

A estrutura de nome *Number* apresenta um índice natural e armazena dados do tipo natural. Esta estrutura armazena o número de entradas para cada índice da estrutura *MDBFlowEntry*, isto é, para cada endereço IP origem. Assim, para a tabela hash exemplo da tabela 3.3, a estrutura *Number* armazenaria : (1:5), (2:4), (3:3), ... Este array facilita a busca dos dados. Se o simulador estiver procurando pelo fluxo (1,4), acessando a estrutura

Number, ele encontra o valor 5. O simulador então fará uma busca sequencial até a posição 5, ou até que encontre o valor 4.

O procedimento *FlowAnalyzer*, figura 3.25, sofreu a adição de quatro variáveis auxiliares : *Add*, *Last*, *Aux4* e *Ant* e, da variável *Biggest1*, que armazena o maior índice da tabela de fluxos candidatos. A variável *Record* é agora declarada como do tipo *MDBFlowEntry*. A variável *Quant* do tipo *Number*. As variáveis *Resp*, *Aux2* e *DBIndex* da versão 1 foram descartadas, figura 3.7.

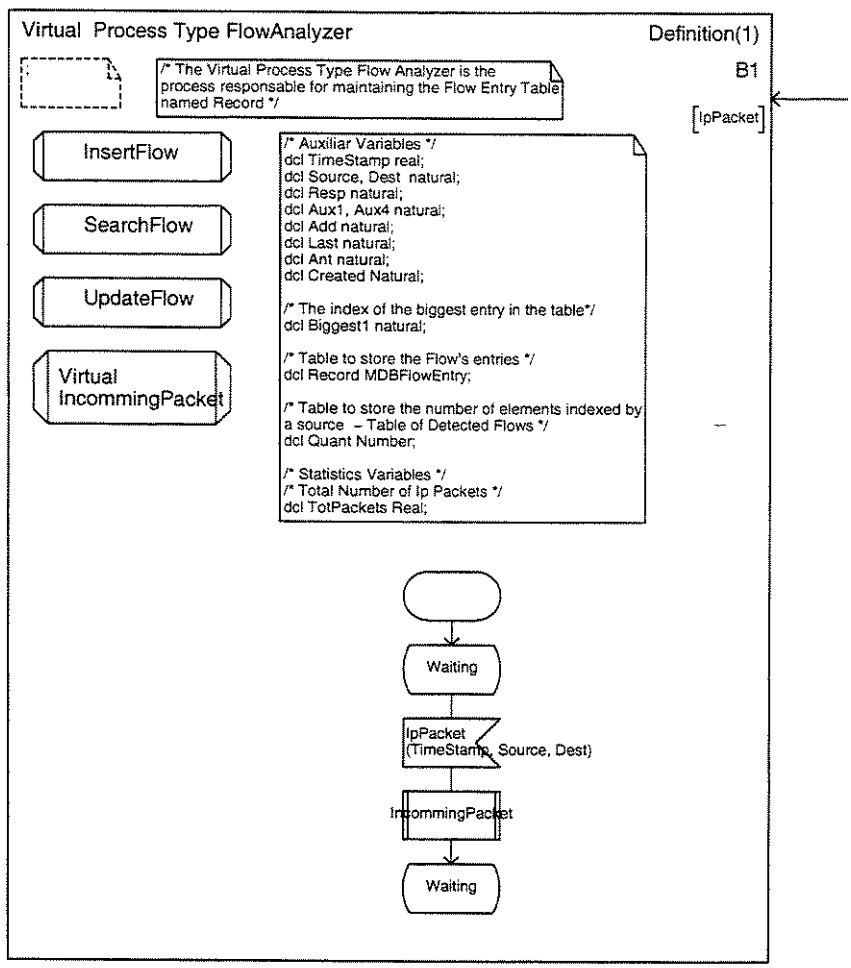

Figura 3.25: Processo FlowAnalyzer (Pacote PackIpS1_1 / Versão 1.1)

Os procedimentos *SearchFlow*, *InsertFlow* e *UpdateFlow* das figuras 3.26, 3.27 e 3.28, continuam tendo o mesmo comportamento funcional que os procedimentos das figuras 3.9, 3.10 e 3.11, mas foram modificados para trabalharem com o acesso à tabela hash.

No procedimento *InsertFlow*, figura 3.27, o primeiro índice da tabela *Record*, a variável auxiliar *Aux1*, recebe o valor do endereço IP origem do fluxo a ser inserido. Uma procura sequencial é feita começando-se na posição 1 (*Add = 1*), até encontrar-se o primeiro maior elemento do que o endereço IP destino a ser inserido. Deste modo, o novo fluxo é inserido na ordem crescente no vetor indexado pelo valor de seu endereço origem. Caso ele não seja inserido na última posição, um *shift-right* é realizado para que ele seja inserido na posição correta.

O procedimento *SearchFlow*, figura 3.26, realiza a procura do fluxo, somente no vetor que tem como índice, o valor do seu endereço IP origem. A procura é feita começando-se pelo valor 1 (*Aux1 :=1*), até que se encontre uma estrutura que possua o valor do endereço IP destino, ou se alcance o final do vetor (*Last := Quant(Source)*).

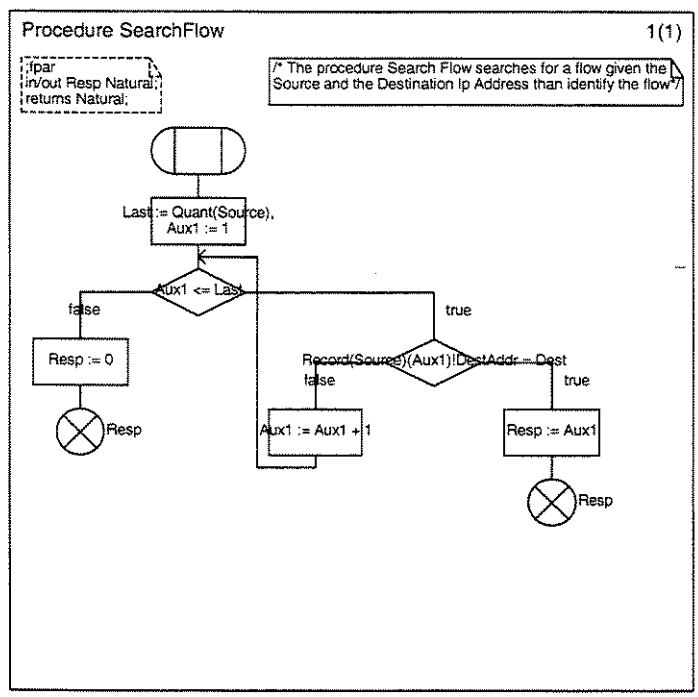

Figura 3.26: Procedimento SearchFlow (Pacote PackIpS1_1/ Versão 1.1)

O procedimento *UpdateFlow*, figura 3.28, atualiza os valores associados a um fluxo dado seu identificador de fluxo. O status do fluxo é atualizado para A (*Active*) e, a variável *Aux4* é incrementada de uma unidade caso o status anterior do fluxo for S (*Switched*). A variável *Aux4* representa o número de fluxos que haviam sido comutados, tornaram-se inativos e voltaram, num determinado instante, a transmitir pacotes.

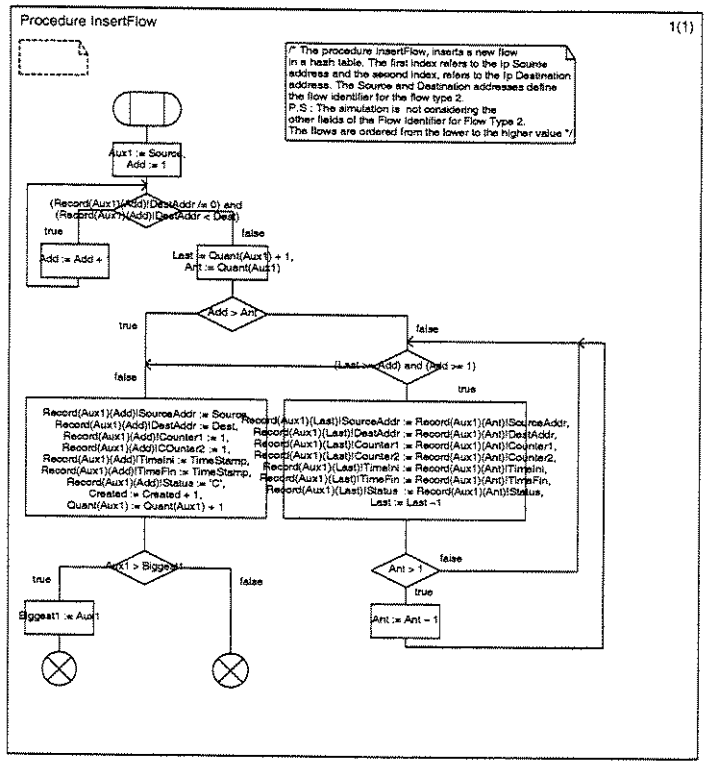

Figura 3.27: Procedimento InsertFlow (Pacote PackIpS1_1 / Versão 1.1)

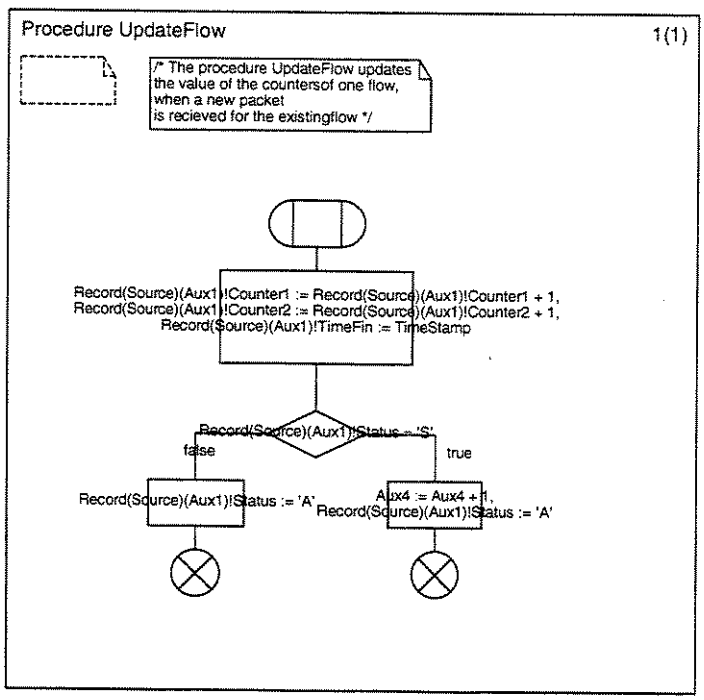

Figura 3.28: Procedimento UpdateFlow (Pacote PackIpS1_1/ Versão 1.1)

3.5 Especificação do Classificador de Fluxo X/Y (Versão 2.1)

Da mesma forma que a versão 1 é responsável pela tabela de fluxos detectados, a versão 2 é responsável pela manipulação da tabela de fluxos comutados. A estrutura da tabela *Label* da versão 2, também possuia um acesso sequencial e contribuia para gerar tempos significativos de simulação.

Com o objetivo de se melhorar o tempo de simulação, as estruturas de dados declaradas na versão 2, também foram alteradas resultando em uma versão 2.1. Da mesma forma que a alteração da versão 1 para a versão 1.1, o modelo funcional da versão 2 não foi alterado na versão 2.1, como pode ser observado comparando-se as figuras 3.1 e 3.2. Os pacotes *PackIpS2* e *PackIpS2_1* possuem exatamente a mesma estrutura SDL, com exceção do procedimento *InactiveFlow* que é acrescentado no processo *FlowAnalyzer* da versão 2.1.

O pacote *PackIpS2* sofreu alterações, resultando no pacote *PackIpS2_1*, somente para se modificar a estrutura de dados que armazenava os dados sobre os fluxos comutados, da mesma forma que a tabela de fluxos detectados foi modificada da versão 1 para a versão 1.1.

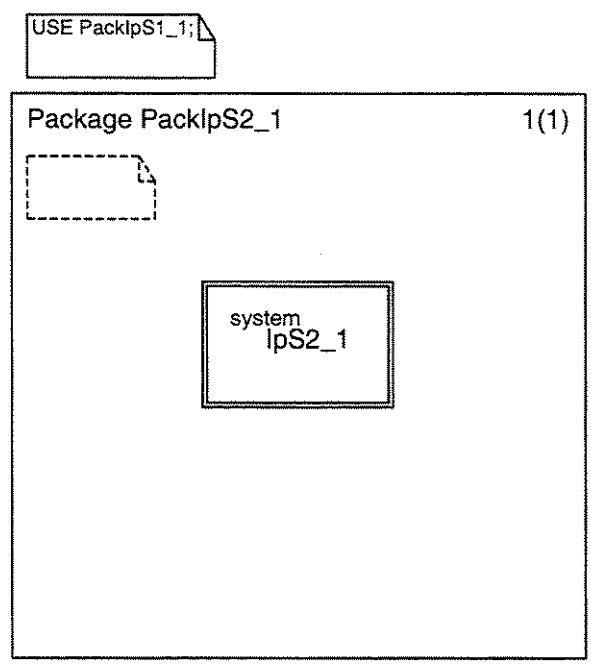

Figura 3.29: Pacote PackIpS2_1 / Versão 2.1

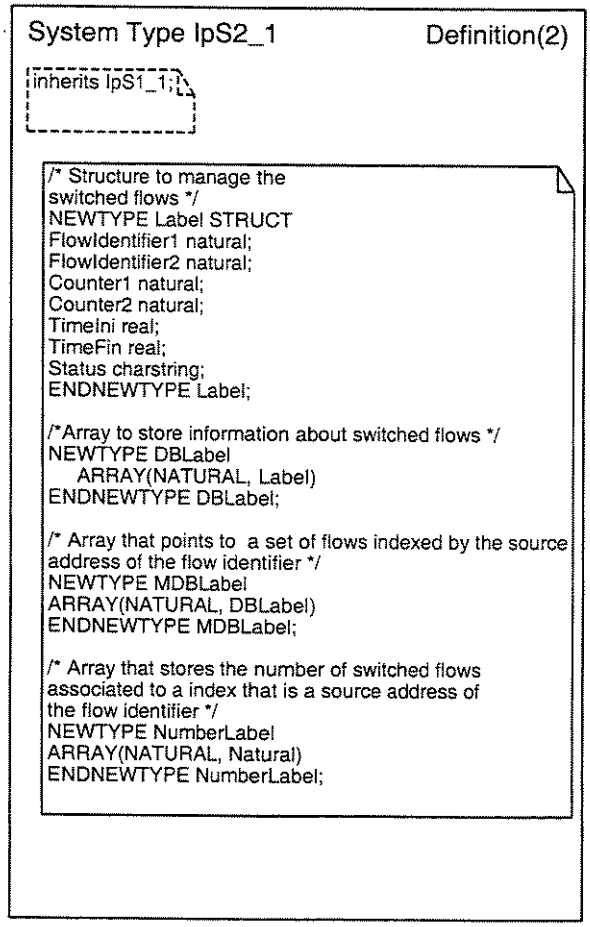

Figura 3.30: Definição do Sistema IpS2_1 (Pacote PackIpS2_1 / Versão 2.1)

A seção 3.5 descreve as alterações realizadas nas estruturas de dados e procedimentos, que foram alterados na versão 2 resultando na versão 2.1. Os blocos, processos e procedimentos que não forem apresentados nesta seção, não sofreram modificações. A versão 2.1 não é herdada da versão 2, mas sim uma modificação da versão 2.

O pacote *PackIpS2* foi renomeado para pacote *PackIpS2_1*, figura 3.29. O sistema *IpS2* também foi renomeado para sistema *IpS2_1* e, sofreu alterações na estrutura de dados que armazena informações sobre os fluxos comutados, como mostrado na figura 3.30.

A figura 3.30 mostra a definição do sistema *IpS2_1*. Na estrutura *Label*, o campo *Status* foi acrescentado com o objetivo de se armazenar o status de um fluxo na tabela de fluxos comutados. Os valores que podem ser atribuídos ao campo status para fluxos comutados, são os mesmos valores atribuídos ao status dos fluxos detectados, com exceção do valor 'S' (*Switched*).

Uma nova estrutura de nome *MDBLabel* é criada. Ela é um vetor de estruturas do tipo *DBLabel*, que por sua vez é um vetor de estruturas *Label*. O endereço IP origem é

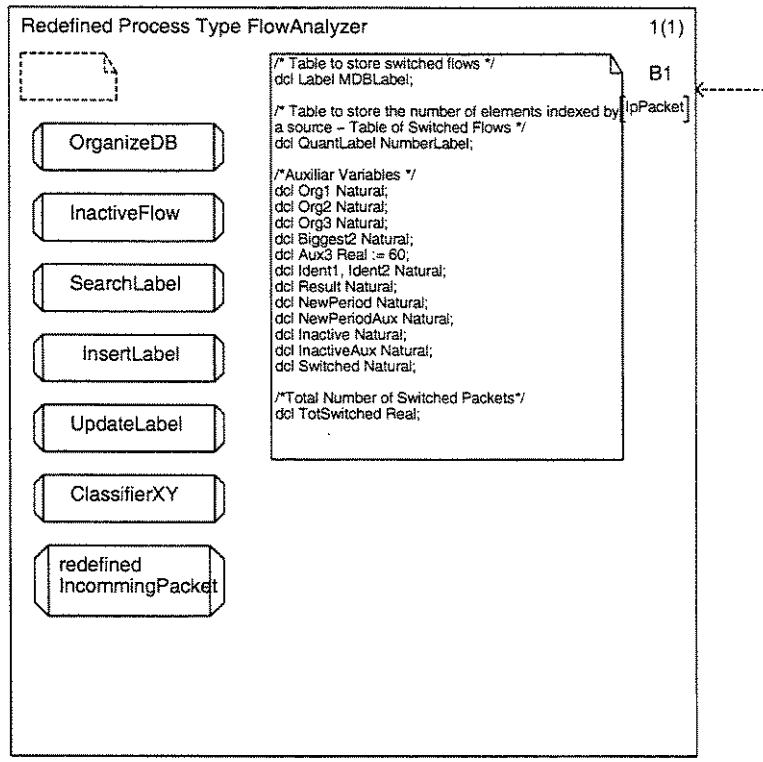

Figura 3.31: Processo FlowAnalyzer (Bloco FlowClass / Versão 2.1)

um índice do tipo natural para o vetor *MDBLabel* e, o endereço IP destino serve de índice para a inserção ordenada no vetor *DBLabel*.

A estrutura de nome *NumberLabel* apresenta um índice natural e armazena dados do tipo natural. Esta estrutura armazena o número de entradas para cada índice da estrutura *MDBLabel* e, tem a mesma função da estrutura *Number* para os fluxos candidatos da versão 1.1.

No processo *FlowAnalyzer* redefinido, figura 3.31 três novas variáveis auxiliares foram definidas (*Org1*, *Org2* e *Org3*) , assim como a variável *Biggest2* que armazena o valor do maior índice da tabela de fluxos comutados. As variáveis *DBIndex2* e *Aux* declaradas na versão 2 da figura 3.16 foram descartadas.

A variável *Label* é agora declarada como do tipo *MDBLabel*. E a variável *QuantLabel* do tipo *NumberLabel* é adicionada.

Algumas variáveis que guardam resultados estatísticos foram criadas como *NewPeriod* que, armazena o número total de vezes que, fluxos da tabela de fluxos candidatos foram reinicializados, para serem analisados em um novo período. Isto significa que, se um fluxo candidato passado Z segundos não atingiu os parâmetros X e Y do classificador de fluxo, ele então será reinicializado. A variável *NewPeriodAux* armazena o número de fluxos que foram reinicializados durante a última verificação de tabela e, não durante todo o tempo de simulação.

A variável *Inactive* armazena o número total de vezes que, fluxos da tabela de fluxos

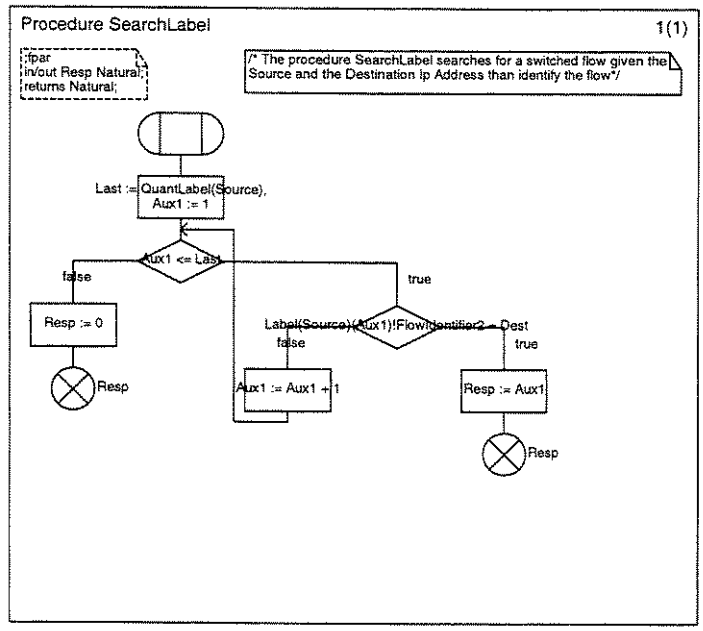

Figura 3.32: Procedimento SearchLabel (Processo FlowAnalyzer / Versão 2.1)

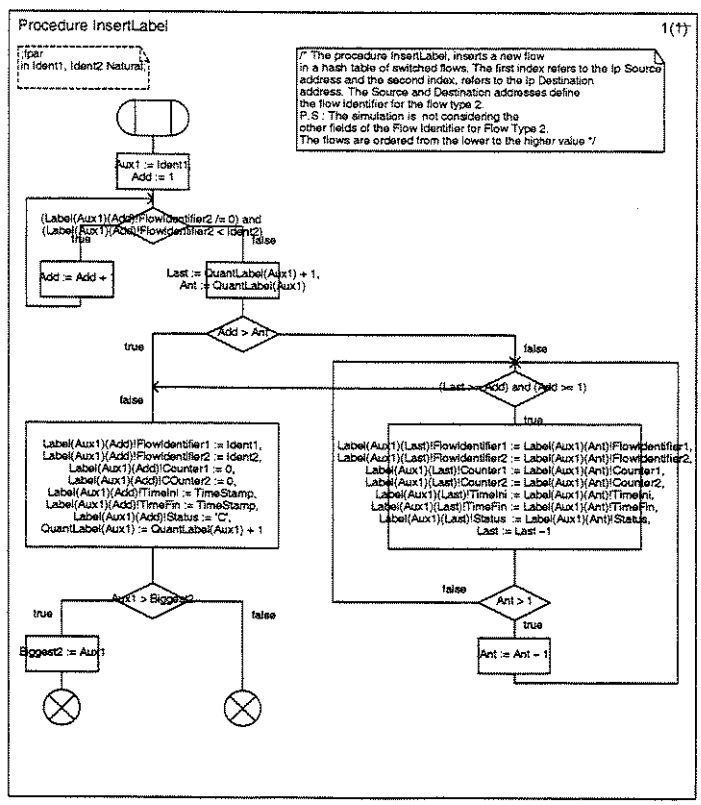

Figura 3.33: Procedimento InsertLabel (Processo FlowAnalyzer / Versão 2.1)

comutados foram considerados inativos. Fluxos comutados são considerados inativos se permanecerem Z segundos sem comutar pacotes. A variável *InactiveAux* armazena o número de fluxos comutados que foram considerados inativos durante a última verificação da tabela de fluxos comutados. E finalmente a variável *Switched* que armazena o número total de fluxos selecionados para a comutação.

O novo procedimento criado de nome *InactiveFlow* da figura 3.34, tem por função remover da tabela de fluxos comutados, aqueles fluxos que foram considerados inativos. O fluxo considerado inativo é removido e o restante dos fluxos de mesmo índice e que estão localizados após o fluxo removido, sofrem um deslocamento à esquerda para a reorganização da tabela. Desta forma, se um fluxo foi considerado inativo e começar a transmitir pacotes novamente, ele tem que ser selecionado mais uma vez pelo classificador de fluxos para ser considerado bom para a comutação de novo.

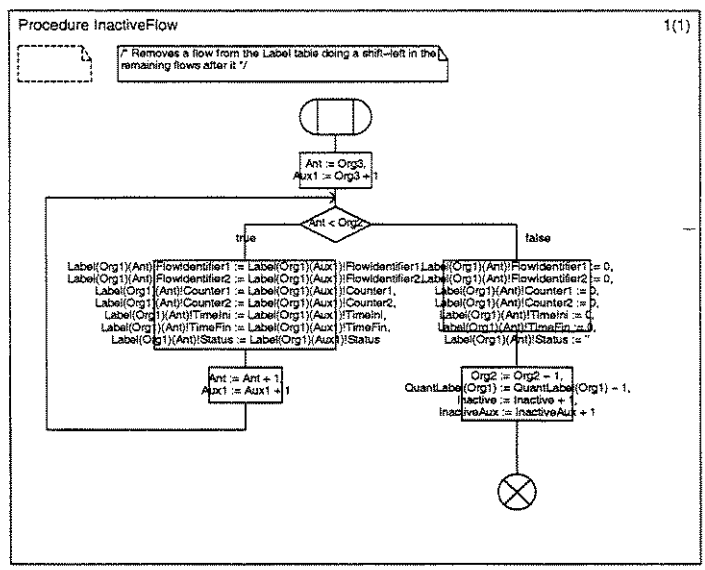

Figura 3.34: Procedimento InactiveFlow4 (Processo FlowAnalyzer / Versão 2.1)

Os procedimentos *SearchLabel* da figura 3.32, *InsertLabel* da figura 3.33 e, o procedimento *UpdateLabel* da figura 3.35, foram alterados da mesma forma que os procedimentos de manutenção da tabela de fluxos candidatos *SearchFlow*, *InsertFlow* e *UpdateFlow* da versão 1.1, para a manipulação da nova estrutura de dados *Label*.

O procedimento *ClassifierXY* foi alterado para acessar os dados da nova estrutura como mostrado na figura 3.36, assim como o procedimento *OrganizeDB*, figura 3.37. As alterações destes dois procedimentos dizem respeito somente ao acesso dos dados, isto é, a maneira de se acessar os dados de acordo com a nova estrutura que pede dois índices ao invés de apenas 1, figuras 3.22 e 3.17.

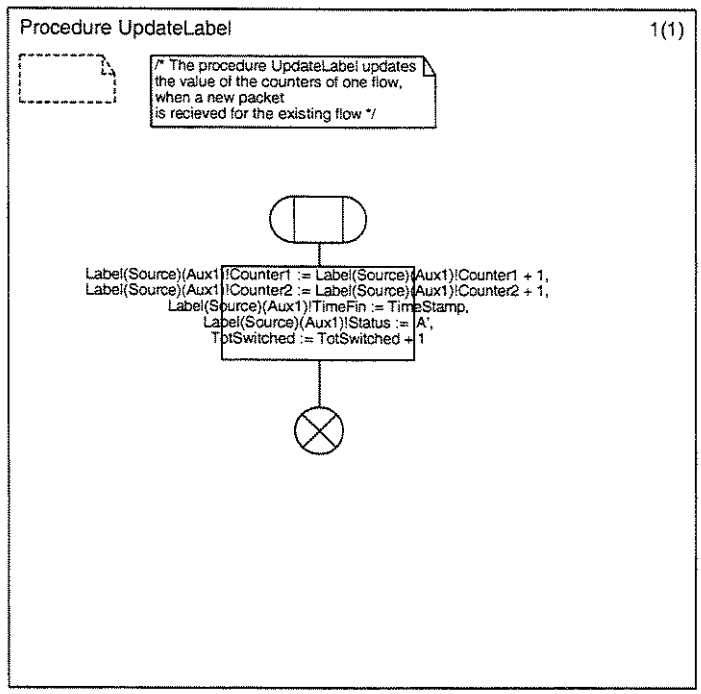

Figura 3.35: Procedimento UpdateLabel (Processo FlowAnalyzer / Versão 2.1)

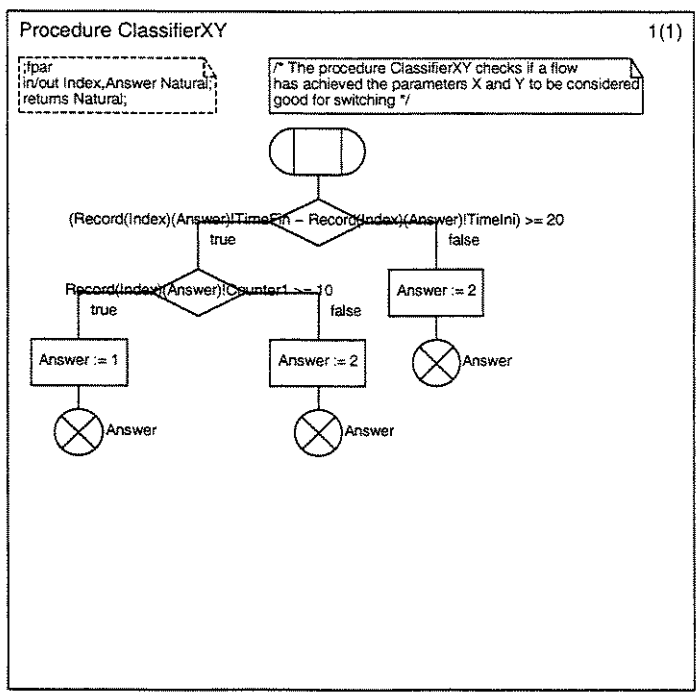

Figura 3.36: Procedimento ClassifierXY (Processo FlowAnalyzer / Versão 2.1)

Podemos obter um exemplo, comparando-se o procedimento *OrganizeDB* das versões 2 e 2.1. Podemos verificar a mudança da estrutura e o acesso aos dados das tabelas *Label* e *Record*. Na versão 2, figura 3.17 as estruturas *Label* e *Record* só utilizavam 1 índice (*Aux*). Já na versão 2.1, figura 3.37, as mesmas estruturas utilizam dois índices, *Org1* e *Org3*. Além disso, o procedimento *OrganizeDB* agora chama o procedimento *InactiveFlow* ao invés de reinicializar a estrutura *Label*.

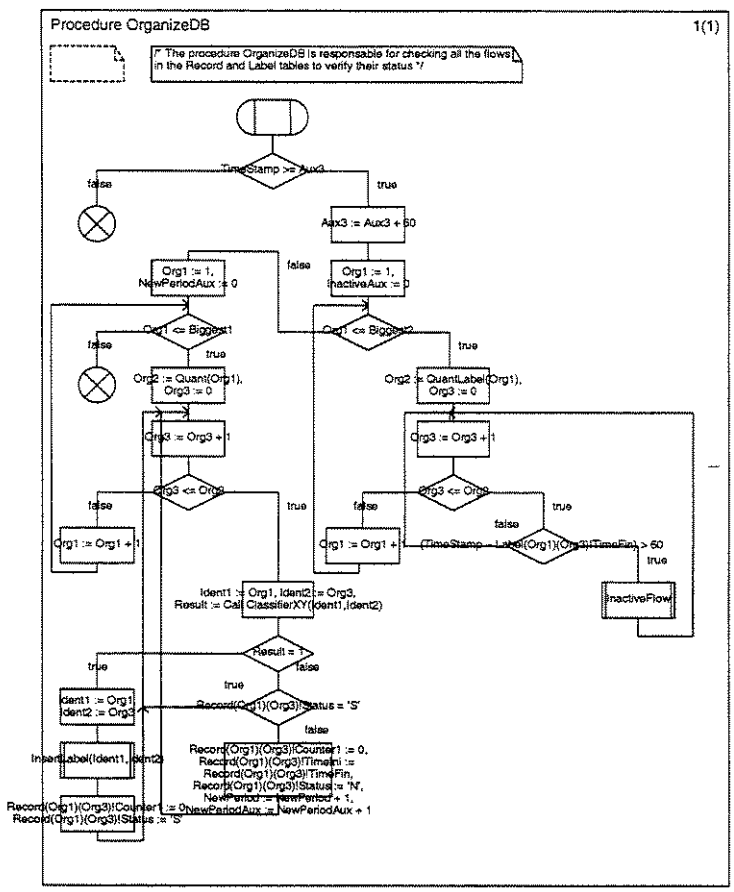

Figura 3.37: Procedimento OrganizeDB (Processo FlowAnalyzer / Versão 2.1)

3.6 MSCs de Simulações da Versão 2.1

Esta subseção apresenta os diagramas MSC (*Message Sequence Chart*), gerados em simulações de algumas situações específicas da versão 2.1. Escolheu-se a versão 2.1 por ser a versão de especificação mais completa e mais eficiente.

Os diagramas MSC representam graficamente o comportamento funcional do sistema em relação à troca de sinais entre o ambiente externo e entre blocos do sistema, assim como as ações tomadas pelo sistema quando do acontecimento de um evento, como a chegada de um sinal.

As cinco situações específicas apresentadas nesta seção, tem por objetivo analisar o comportamento funcional do classificador de fluxo diante de chegadas de pacotes IP no controlador do IpSwitch.

A primeira situação, figura 3.38, analisa a chegada de um pacote IP pertencente a um fluxo que ainda não foi detectado pelo classificador de fluxo.

A segunda situação, figura 3.39, analisa a chegada de um pacote IP pertencente ao mesmo fluxo que foi detectado na primeira situação, mas que ainda não foi selecionado para a comutação.

A terceira situação, figura 3.40, analisa a chegada de um pacote IP pertencente ao fluxo detectado na primeira situação, mas que será selecionado para comutação com a chegada deste novo pacote.

A quarta situação, figura 3.41, analisa a chegada de um pacote IP pertencente ao fluxo detectado na primeira situação e, que foi selecionado para comutação na terceira situação.

E finalmente, a quinta situação, figura 3.42, ilustra o caso do mesmo fluxo, já comutado, ser considerado inativo.

As cinco situações e seus respectivos diagramas MSCs são descritas nas cinco subseções a seguir.

3.6.1 Primeira Situação : Chegada de Pacote de um Fluxo não Detectado

A primeira situação de simulação ilustra o caso de chegada de um pacote IP no controlador do IpSwitch, e que é pertencente a um fluxo que ainda não foi detectado pelo classificador de fluxo.

Isto significa que, este é o primeiro pacote de um novo fluxo, isto é, o fluxo não pertence à tabela de fluxos detectados e nem à tabela de fluxos comutados. É o primeiro pacote IP deste fluxo identificado pelo seu endereço origem e destino, que chega ao controlador.

Diante desta situação, o controlador deve detectar a não existência de um fluxo para este pacote IP e, criar então uma nova entrada na tabela de fluxos detectados considerando como identificador de fluxo, os endereços IP origem e destino do pacote recebido.

De acordo com a figura 3.38, podemos observar o comportamento funcional do processo *Analyzer* pertencente ao bloco *FlowClass*, diante do recebimento deste pacote IP.

Inicialmente o sistema encontra-se no estado *Waiting* pronto para atender a chegada de um novo pacote IP. Em um certo instante da simulação, ocorre o envio de um sinal

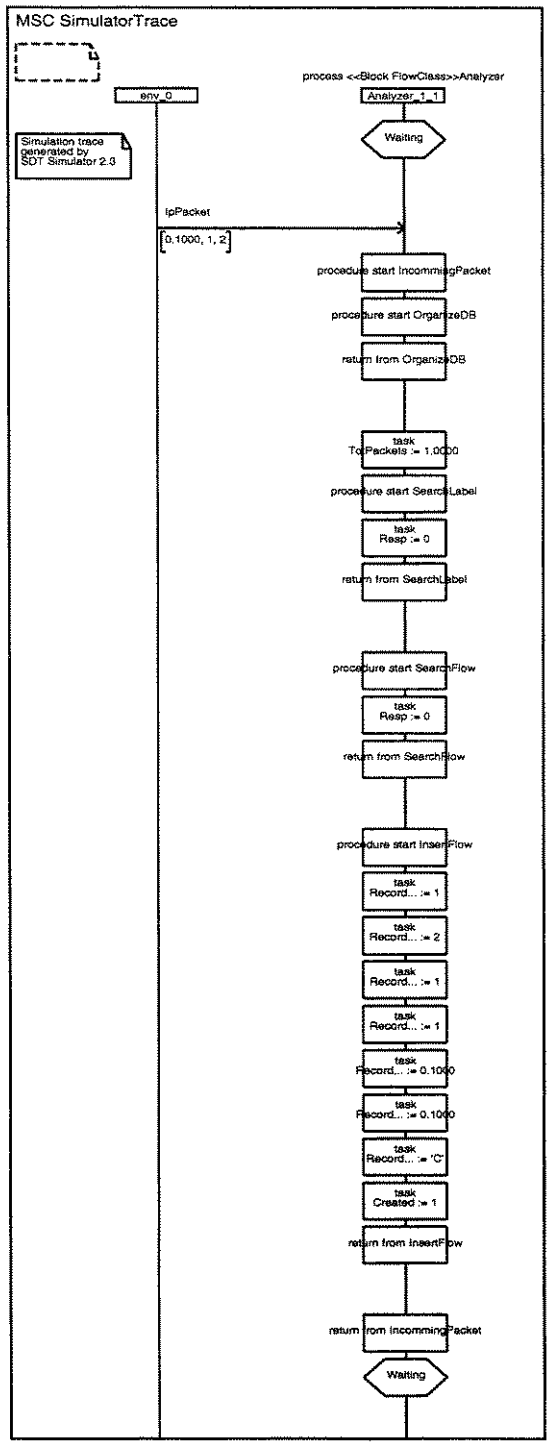

Figura 3.38: MSC da Primeira Situação de Simulação/ Versão 2.1

IpPacket do meio externo para o bloco *FlowClass*. O envio deste sinal representa a chegada do pacote IP no controlador. O sinal *IpPacket* possui três parâmetros [0.1000, 1, 2] como mostrado na figura 3.38. O primeiro parâmetro representa o *TimeStamp* do pacote IP, neste caso o valor é 0.1000. O segundo e o terceiro parâmetros representam os endereços origem e destino do pacote IP respectivamente, neste caso 1 e 2.

A ação tomada pelo sistema é passar o sinal ao processo *Analyzer*, figura 3.31, que ativa o procedimento *IncommingPacket* , figura 3.21. O procedimento *IncommingPacket*, ativa o processo *OrganizeDB*, figura 3.37 que não é executado pois o *TimeStamp* não é um múltiplo de 60. Neste caso as tabelas são verificadas de 60 em 60 segundos. Desta forma, o sistema retorna do processo *OrganizeDB*.

O número total de pacotes IP recebidos é incrementado, neste caso específico ele era o primeiro pacote IP a ser recebido pelo sistema por isso a variável *TotPackets* recebeu o valor 1.

Em seguida, o sistema chama o procedimento *SearchLabel*, figura 3.32, para verificar se o pacote pertence a um fluxo que foi selecionado para comutação. A procura não obteve sucesso, a variável *Resp* recebeu o valor 0 (indica que o fluxo não foi encontrado na tabela) e, o sistema retorna do procedimento *SearchLabel*.

Como o fluxo não foi encontrado na tabela de fluxos comutados, o sistema vai então procurar pelo fluxo na tabela de fluxos detectados chamando o procedimento *SearchFlow*, figura 3.26. Mais uma vez a procura não obteve sucesso, a variável *Resp* recebeu o valor 0 e, o sistema retorna da execução do procedimento *SearchFlow*.

Como o fluxo não foi encontrado em nenhuma das duas tabelas, o sistema decide inserí-lo como um novo fluxo na tabela de fluxos detectados e, chama então o procedimento *InsertFlow*, figura 3.27.

Pode-se observar pela sequência de quadros no MSC da figura 3.38, que os campos da tabela *Record* são preenchidos com os valores 1, 2, 1, 1, 0.1000, 0.1000 e C. Estes valores representam respectivamente o endereço IP origem, o endereço IP destino, contador auxiliar, contador auxiliar, o instante da chegada do primeiro pacote deste fluxo, o instante da chegada do último pacote deste fluxo e, a letra C que inicializa o status do fluxo, fluxo criado recentemente.

Os valores 0.1000, 1 e 2 são os valores recebidos pelo sinal IpPacket. Os contadores auxiliares são inicializados com o valor 1, pois este é o primeiro pacote recebido neste fluxo, assim como os tempos inicial e final possuem o mesmo valor, 0.1000.

A variável *Created* é incrementada de uma unidade, neste caso é o primeiro fluxo criado no sistema. O sistema retorna do procedimento *InsertFlow*, retorna do procedimento *IncommingPacket* e, volta ao estado *Waiting* estando pronto para receber o próximo pacote IP.

3.6.2 Segunda Situação : Chegada de um Pacote IP de um Fluxo Detectado

A segunda situação ilustra a chegada de um pacote IP pertence a um fluxo que já foi detectado pelo controlador.

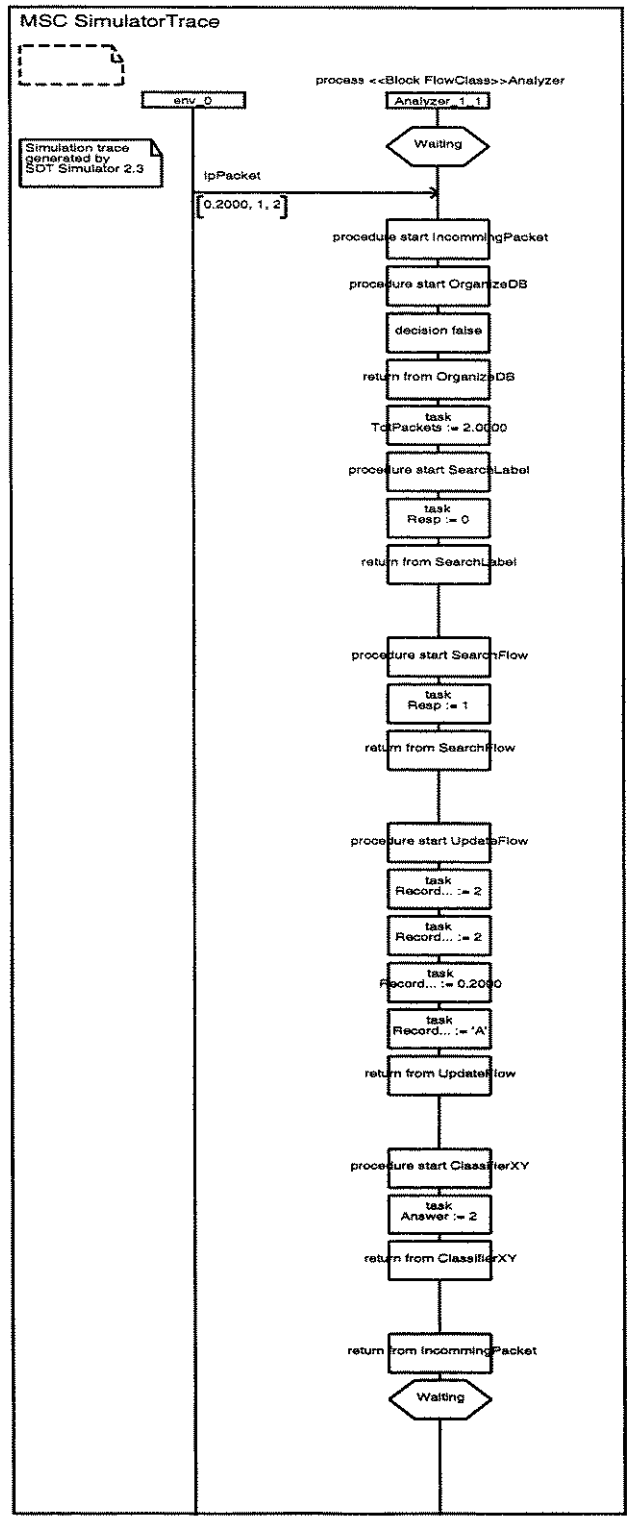

Figura 3.39: MSC da Segunda Situação de Simulação/ Versão 2.1

Nesta situação o controlador deve procurar pelo fluxo através de seu identificador, atualizar os dados do fluxo e então, verificar se o fluxo já apresenta características favoráveis para ser selecionado para comutação, isto é, se o fluxo já atende aos parâmetros X e Y. De acordo com a figura 3.39 podemos observar as ações realizadas pelo sistema diante desta situação.

Inicialmente o sistema mantém-se no estado *Waiting* aguardando pela chegada de um sinal. Em um determinado instante da simulação, um sinal *IpPacket* com os parâmetros [0.2000, 1, 2], é enviado do ambiente externo para o bloco *FlowClass*. O sinal é então passado para o processo *Analyzer* que ao consumí-lo, faz uma chamada ao procedimento *IncommingPacket*.

O procedimento *IncommingPacket* por sua vez, faz uma chamada ao procedimento *OrganizeDB* que não é executado pois o *TimeStamp* do pacote recebido não é um múltiplo de 60 segundos.

O sistema retorna da chamada ao procedimento *OrganizeDB*. A variável *TotPacket* é incrementada de uma unidade, neste caso recebeu o valor 2. Isto significa que este é o segundo pacote, ou seja, o segundo sinal *IpPacket* recebido do meio externo.

O sistema ativa na sequência o procedimento *SearchLabel* para verificar se o pacote IP recebido pertence a um fluxo que já foi selecionado para a comutação. A busca não obtém sucesso, a variável *Resp* recebe o valor 0 indicando que o fluxo não foi encontrado na tabela de fluxos comutados e, o sistema retorna da chamada ao procedimento *SearchLabel*.

Desta forma, o procedimento *SearchFlow* é chamado para agora ser verificado se o pacote IP recebido, pertence à um fluxo que já foi detectado pelo controlador, mas que não foi selecionado para comutação. A busca à tabela de fluxos candidatos se dá com sucesso, a variável *Resp* recebe o valor 1 indicando que o fluxo foi encontrado e seu primeiro índice na tabela é o valor 1. Como os fluxos são indexados pelos endereços origem primeiramente e depois pelo endereço destino, o primeiro índice do fluxo procurado é o valor 1 proveniente do endereço IP origem 1.

O sistema retorna da execução do procedimento *SearchFlow*. Como o pacote recebido pertence a um fluxo existente, seus dados são atualizados através da execução do procedimento *UpdateFlow*. Seguindo-se os quadros de ações do diagrama da figura 3.39, pode-se observar que os campos da estrutura da tabela *Record* foram atualizados com os valores 2, 2, 0.2000 e A. Os dois primeiros valores 2, são os contadores auxiliares que foram incrementados de uma unidade. O valor 0.2000 é atribuído ao campo que armazena o tempo de chegada do último pacote do fluxo e, finalmente a letra A, atualiza o status do fluxo para *Active*. O sistema retorna da chamada ao procedimento *UpdateFlow*.

O procedimento *ClassifierXY* é chamado com o objetivo de verificar se os parâmetros X e Y foram atingidos pelo fluxo que acabou de receber um novo pacote. O fluxo não atingiu os parâmetros, que no caso desta simulação é de X igual a 10 pacotes e Y igual a 20 segundos. A variável *Answer* recebe o valor 2 que indica fluxo não selecionado e, o sistema retorna da chamada do procedimento *ClassifierXY*.

O sistema retorna do procedimento *IncommmingPacket* e volta ao estado *Waiting* estando pronto para receber um novo pacote.

3.6.3 Terceira Situação : Chegada de um Pacote de um Fluxo que será selecionado para a Comutação

A terceira situação ilustra o caso da chegada de um pacote IP pertencente a um fluxo que já foi detectado pelo controlador e, que atingiu os parâmetros X e Y com a chegada deste novo pacote IP.

O controlador deve então desativar o fluxo na tabela de fluxos detectados e inserí-lo na tabela de fluxos comutados. A figura 3.40 descreve as ações tomadas pelo sistema sequencialmente.

Primeiramente o sistema encontra-se no estado *Waiting* aguardando pelo envio de um sinal *IpPacket*. Num determinado instante da simulação houve o envio do sinal *IpPacket* com os parâmetros [21.00, 1, 2] do ambiente externo para o bloco *FlowClass*. O sinal é recebido pelo processo *Analyzer* que invoca o procedimento *IncommingPacket*. O procedimento *OrganizeDB* é chamado mas retorna como nos casos anteriores.

O número total de pacotes recebidos pelo sistema, *TotPackets*, é incrementado de uma unidade. O procedimento *SearchLabel* é chamado para verificar se o pacote recebido pertence à um fluxo já selecionado para comutação. A busca não obtém sucesso, a variável *Resp* recebe o valor 0 indicando que o fluxo não foi encontrado e, o sistema retorna da chamada ao procedimento *SearchLabel*.

O procedimento *SearchFlow* é então chamado para verificar se o pacote recebido pertence à um fluxo já detectado pelo controlador. A busca se dá com sucesso, a variável *Resp* recebe o valor 1 indicando que o fluxo foi encontrado na tabela e que o primeiro índice é o valor 1. O sistema retorna da chamada ao procedimento *SearchFlow*.

Na sequência é chamado o procedimento *UpdateFlow* que atualiza os valores do fluxo ao qual o pacote recebido pertence. Pode-se observar na figura 3.40 a sequência de quadros de ações que atualiza os campos da tabela *Record*. Os valores atualizados são 15, 15, 21.0000 e A. Os dois primeiros números são a atualização dos contadores auxiliares acrescidos de uma unidade. O valor 21.0000 é o valor do *TimeStamp* recebido do sinal *IpPacket* e, atualiza o valor do tempo de chegada do último pacote do fluxo. A letra A atualiza o status do fluxo e indica que o fluxo mantém-se ativo. O sistema então retorna da chamada ao procedimento *UpdateFlow*.

O procedimento *ClassifierXY* é chamado para verificar se o fluxo que acabou de receber um pacote IP já atingiu os parâmetros X e Y, 10 pacotes e 20 segundos. Neste caso, o procedimento foi executado com sucesso, a variável *Answer* recebeu o valor 1 que indica que o fluxo foi selecionado para comutação. O sistema retorna da chamada ao procedimento *ClassifierXY*.

O número de fluxos comutados, variável *Switched*, é incrementado de uma unidade. Neste caso, este é o primeiro fluxo do sistema a ser selecionado para comutação.

Como o fluxo foi selecionado para comutação ele deve ser inserido na tabela de fluxos comutados. Isto é feito com a chamada ao procedimento *InsertLabel*. A sequência de quadros na figura 3.40 mostra a inicialização do fluxo na tabela *Label* com os valores 1, 2, 0, 0, 21.00, 21.00 e C.

Os valores 1 e 2 são os identificadores do fluxo, o endereço IP origem e o endereço

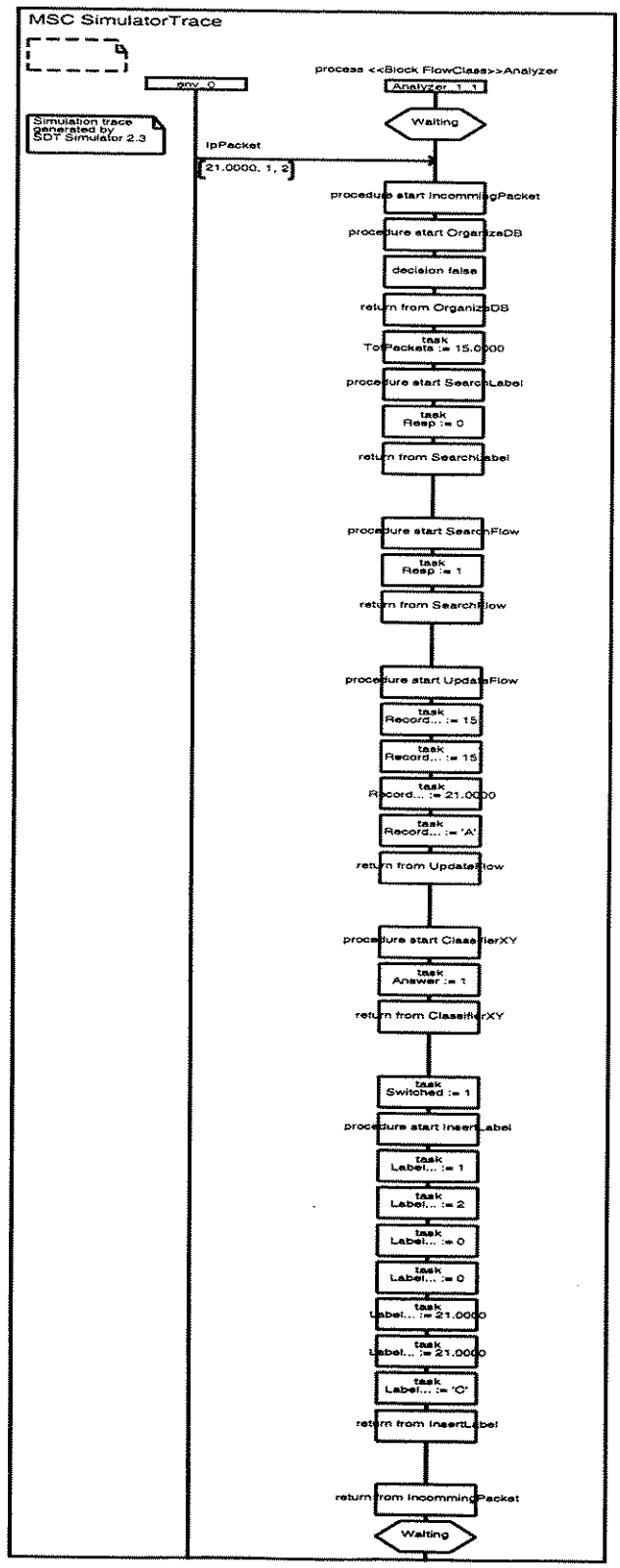

Figura 3.40: MSC da Terceira Situação de Simulação/ Versão 2.1

IP destino. Os valores 0 e 0 incializam os contadores auxiliares. Os contadores são inicializados com 0 pois nenhum pacote para este fluxo foi comutado até o momento. O tempo de recebimento do primeiro pacote e o tempo de recebimento do último pacote são inicializados com o *TimeStamp* do pacote recebido. Isto significa que nenhum pacote foi comutado, porém que o fluxo foi selecionado para comutação quando o pacote de *TimeStamp* 21.0000 foi recebido. E finalmente a letra C inicializa o status do fluxo indicando que o fluxo foi criado na tabela de fluxos comutados. O sistema retorna da chamada ao procedimento *InsertLabel.*

O sistema também retorna da chamada ao procedimento *IncommingPacket* e volta ao estado *Waiting* estando pronto para receber o próximo sinal *IpPacket.*

3.6.4 Quarta Situação : Chegada de Pacote pertencente a um Fluxo Comutado

A quarta situação ilustra o caso da chegada de um pacote IP que pertence a um fluxo que já foi selecionado para comutação.

O controlador deve procurar pelo fluxo na tabela de fluxos comutados através do identificador de fluxo e então atualizar os seus dados. A figura 3.41 mostra o diagrama MSC com estados e ações do sistema.

Primeiramente o sistema encontra-se no estado *Waiting* aguardando pelo envio de um sinal. Num determinado instante da simulação, o sinal *IpPacket* com os parâmetros [21.1000, 1, 2] é enviado pelo ambiente externo para o bloco *FlowClass.*

O sinal é consumido pelo processo *Analyzer* que invoca o procedimento *Incomming-Packet.* O procedimento *IncommingPacket* chama o procedimento *OrganizeDB* que não é executado pois o *TimeStamp* do pacote recebido não é um múltiplo de 60 segundos. Desta forma o sistema retorna da chamada ao procedimento *OrganizeDB.*

O número total de pacotes IP recebidos, variável *TotPackets* é incrementado de uma unidade, neste caso o sistema havia recebido 15 pacotes até a chegada deste novo pacote.

O sistema ativa o procedimento *SearchLabel* para verificar se o pacote recebido pertence à um fluxo que já foi selecionado para comutação. A busca se dá com sucesso. O fluxo foi encontrado e a variável *Resp* recebe o valor do primeiro índice para o fluxo, neste caso o valor 1. O sistema retorna da chamada ao procedimento *SearchLabel.*

Como o fluxo foi encontrado na tabela de fluxos comutados, os seus dados devem ser atualizados. Para isso o sistema chama o procedimento *UpdateLabel.* Seguindo-se os quadros da figura 3.41 pode-se observar que os dados da estrutura *Label* foram atualizados com os valores 1, 1, 21.1000 e A. Os dois primeiros valores incrementam os contadores auxiliares de uma unidade. Como o valor é 1, significa que este é o primeiro pacote a ser comutado neste fluxo. O valor 21.100 atualiza o campo da tabela que armazena o tempo de chegada do último pacote neste fluxo e, a letra A, atualiza o status do fluxo para *Active.* O sistema retorna da chamada ao procedimento *UpdateLabel.*

O sistema retorna então da chamada ao procedimento *IncommingPacket* e volta para o estado *Waiting* estando pronto para receber o próximo sinal *IpPacket.*

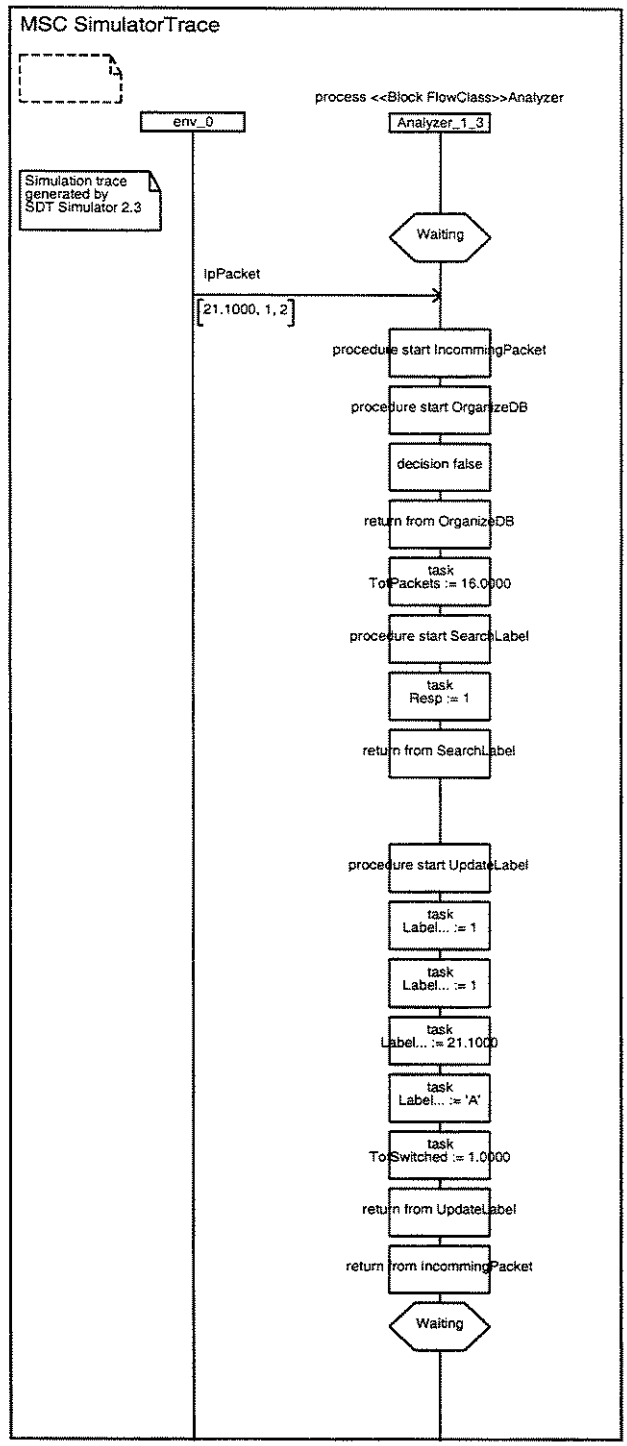

Figura 3.41: MSC da Quarta Situação de Simulação/ Versão 2.1

3.6.5 Quinta Situação : Fluxo Comutado torna-se Inativo e volta a transmitir Pacote

A quinta situação é a simulação do caso onde um fluxo transmite pacotes como fluxo candidato, é selecionado para a comutação, torna-se inativo e volta a transmitir pacotes.

Com a chegada de um pacote com *TimeStamp* superior a 60 segundos, o sistema deve então verificar as tabelas de fluxos comutados e detectados.

Apenas as partes relevantes do MSC serão apresentadas nesta seção, figura 3.42. A tabela 3.5 apresenta o estado das variáveis em cinco passos da simulação que são descritas a seguir. Durante a simulação foram utilizados os parâmetros X=5 pacotes, Y=10 segundos, verificação das tabelas de 60 em 60 segundos e tempo de fluxo inativo de 60 segundos.

De acordo com a tabela 3.5, pode-se observar que no primeiro estado registrado, a tabela de fluxos comutados *Label*, não apresenta fluxos selecionados. O número de fluxos detectados, *Created*, é de 2 e, o total de pacotes transmitidos, *TotPackets*, é de 6.

A tabela de fluxos detectados *Record* apresenta dois fluxos. O primeiro fluxo possui endereço IP origem 1, endereço IP destino 2, transmitiu 5 pacotes, o primeiro pacote no instante 0.1000 e o último pacote no instante 0.5000 e, o estado do fluxo é ativo, A. O segundo fluxo tem endereço IP origem 1, endereço IP destino 3, transmitiu 1 pacote, no instante 0.1000, e seu estado é de fluxo criado, C.

No segundo estado da tabela 3.5 o fluxo detectado de endereço origem 1 e endereço destino 2, recebeu um pacote no instante 11 e foi selecionado para a comutação. O fluxo foi inserido na tabela *Label* onde o endereço IP origem é igual a 1, o endereço IP destino é igual a 2, os contadores são inicializados com o valor 0 já que nenhum pacote foi comutado, os instantes de chegada do primeiro e último pacote, são registrados com o valor do instante em que o fluxo foi selecionado para a comutação (11) e, finalmente o status do fluxo é de criado, C.

O número de fluxos detectados, *Created* continua sendo 2. O número de fluxos comutados, *Switched*, passa a valer 1. Nenhum pacote foi comutado, *TotSwitched*. O número total de pacotes transmitidos, *TotPackets* é de 7. Na tabela *Record* o fluxo que foi comutado, endereço origem 1 e endereço destino 2, tem seu primeiro contador auxiliar reiniciado com o valor 0, o segundo armazena o número de pacotes transmitidos antes de ser comutado, 6 pacotes, o instante da chegada do primeiro e último pacote, 0.1000 e 11.0000 e o status passa para S, *switched*.

No terceiro estado da tabela 3.5 o fluxo da tabela *Label* comuta um pacote no instante 12 e tem seus contadores auxiliares modificados para o valor 1, e o tempo de chegada do último pacote para 12. O número de pacotes comutados, *TotSwitched*, passa a ser 1, o número total de pacotes transmitidos, *TotPackets*, é incrementado de uma unidade.

No quarto estado da tabela 3.5 houve a verificação das tabelas de fluxo comutados e detectados. O fluxo comutado foi considerado inativo e foi removido da tabela. A figura 3.42 descreve as ações tomadas pelo sistema. Inicialmente o sistema encontra-se no estado *Waiting*. Com a chegada do sinal *IpPacket(80.0000,1,3)* o procedimento *IncommingPacket* do processo *Analyzer* é ativado. Como o *TimeStamp* do pacote recebido tem o valor superior a 60 segundos, o procedimento *OrganizeDB* é ativado. A tabela de

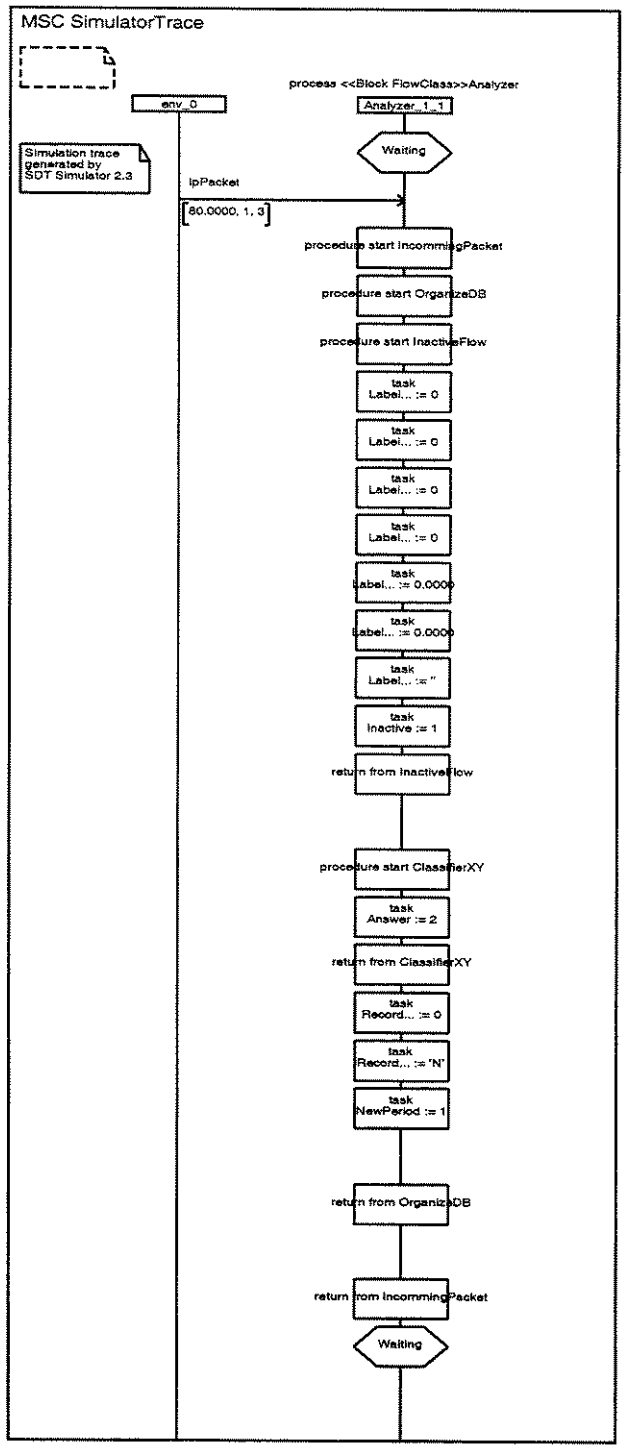

Figura 3.42: MSC da Quinta Situação de Simulação/ Versão 2.1

Estado	Estado das Variáveis
1	Label (MDBLabel) = (others:(. 0, 0, 0, 0, 0.0000, 0.0000, " .)) Created (natural) = 2 TotPackets (real) = 6.0000 Record (MDBFlowEntry) = (1:(. 1, 2, 5, 5, 0.1000, 0.5000, 'A' .)), (2:(. 1, 3, 1, 1, 0.1000, 0.1000, 'C' .))
2	Label (MDBLabel) = (1:(. 1, 2, 0, 0, 11.0000, 11.0000, 'C' .)) :)) :) Created (natural) = 2 Switched (natural) = 1 TotSwitched (real) = 0.000 TotPackets (real) = 7.0000 Record (MDBFlowEntry) = (1:(. 1, 2, 0, 6, 0.1000, 11.0000, 'S' .)), (2:(. 1, 3, 1, 1, 0.1000, 0.1000, 'C' .))
3	Label (MDBLabel) = (1:(. 1, 2, 1, 1, 11.0000, 12.0000, 'A' .)) Created (natural) = 2 Switched (natural) = 1 TotSwitched (real) = 1.0000 TotPackets (real) = 8.0000 Record (MDBFlowEntry) = (1:(. 1, 2, 0, 6, 0.1000, 11.0000, 'S' .)), (2:(. 1, 3, 1, 1, 0.1000, 0.1000, 'C' .))
4	Label (MDBLabel) = (1:(. 0, 0, 0, 0, 0.0000, 0.0000, " .)) NewPeriod (natural) = 1 Inactive (natural) = 1 Created (natural) = 2 Switched (natural) = 1 TotSwitched (real) = 1.0000 TotPackets (real) = 9.0000 Record (MDBFlowEntry) = (1:(. 1, 2, 0, 6, 0.1000, 11.0000, 'S' .)), (2:(. 1, 3, 1, 2, 0.1000, 80.0000, 'A' .))
5	Label (MDBLabel) = (1:(. 0, 0, 0, 0, 0.0000, 0.0000, " .)) Created (natural) = 2 Switched (natural) = 1 Aux4 (natural) = 1 TotSwitched (real) = 1.0000 TotPackets (real) = 10.0000 Record (MDBFlowEntry) = (1:(. 1, 2, 1, 7, 0.1000, 84.0000, 'A' .)), (2:(. 1, 3, 1, 2, 0.1000, 80.0000, 'A' .))

Tabela 3.5: Valores das Variáveis Durante a Simulação da Quinta Situação

fluxos comutados é analisada para se verificar se algum fluxo comutado tornou-se inativo. O procedimento *InactiveFlow* é ativado e identifica que o único fluxo comutado está inativo pois o último pacote foi comutado no instante 12, isto é, ele está a 68 segundos sem comutar pacotes (80-12=68). Os quadros de ações mostram a remoção dos dados do fluxo da tabela *Label* e a variável *inactive* acrescida de uma unidade, 1 fluxo considerado inativo.

O sistema retorna da execução do procedimento *InactiveFlow* e faz a verificação da tabela de fluxos detectados chamando o processo *ClassifierXY* para verificar se algum fluxo detectado atingiu os parâmetros X e Y e não foi comutado. O fluxo de endereço origem 1 e endereço destino 3 é reincializado tendo o primeiro contador zerado e o status modificado para N, como pode ser observado nos quadros de ações da figura 3.42. O sistema retorna da execução do procedimento *OrganizeDB*, faz o tratamento da chegada do pacote [80.0000,1,3] que não é mostrado no MSC. O sistema retorna então do procedimento *IncommingPacket* e volta ao estado *Waiting* pronto para receber o próximo pacote.

O estado das variáveis após esta execução pode ser observada no quarto estado da tabela 3.5. A tabela *Label* teve o fluxo removido. A variável *NewPeriod* registra o valor 1 pois 1 fluxo detectado foi reinicializado. A variável *TotPackets* foi acrescida de uma unidade. E o fluxo de endereço origem 1 e destino 3, foi reinicializado. Seu primeiro contador foi zerado na execução do procedimento *OrganizeDB*, entretanto, recebeu o pacote [80.0000,1,3] e o contador passou para o valor 1 no novo período de análise além de seu statūs voltar para A de ativo.

Finalmente o quinto estado da tabela 3.5 mostra o estado das variáveis quando o fluxo de endereço origem 1 e destino 2, depois de ter sido considerado inativo, volta a transmitir pacotes. Pode-se observar que na tabela *Record* seu primeiro contador é reinicializado com o valor 1, o instante do recebimento do último pacote com o valor 84.0000 e o status atualizado para A, pois tornou-se ativo novamente. A variável *Aux4* registra o número de fluxos considerados inativos que voltaram a transmitir pacotes e a variável *TotPackets* é acrescida de uma unidade.

3.7 Resultados das Simulações

Esta seção apresenta os resultados obtidos de simulações realizadas com as especificações em SDL descritas nas seções 3.4 e 3.5, a Especificação do Analisador de Fluxo (Versão 1.1), e a Especificação do Classificador de Fluxo X/Y (Versão 2.1). Utilizaram-se as versões 3.4 e 3.5 por serem as versões otimizadas.

As simulações foram realizadas com o SDT *Simulator* e foram utilizados como alimentadores do simulador 7 arquivos de *trace* de tráfego de rede Internet (*DEC-PKT-1, DEC-PKT-2, DEC-PKT-3, DEC-PKT-4, LBL-PKT-4, LBL-PKT-5* e *LBL-TCP-3*). Estes traces apresentam-se disponíveis gratuitamente no *Internet Traffic Archive* por ftp em (*ita.ee.lbl.gov/traces*). A tabela 3.6 apresenta as características dos traces utilizados.

Foram tomados apenas os primeiros 5 minutos de cada *trace* nas simulações devido aos elevados tempos de simulação necessários para a obtenção dos resultados. Além disso

foram analisados somente os pacotes TCP de cada trace.

O SDT *Simulator* não é uma ferramenta apropriada para simulações do tipo que requerem arquivos de alimentação do sistema grandes como os arquivos de *trace* utilizado. Em [23] encontra-se um simulador de mesma função desenvolvido na linguagem C e, que permite simulações de *traces* de tráfego IP de uma hora de duração.

Apesar do SDT não ser a ferramenta apropriada, ele foi utilizado não só com o objetivo de se analisar o desempenho do protocolo IpSwitching, mas também com o objetivo de se especificar formalmente o classificador de fluxo. Através da utilização do SDT foi possível construir um analisador de fluxo (*PackIpS1_1*) e, através do conceito de orientação a objetos de reutilização contidos na linguagem SDL92, construir um classificador de fluxo do tipo X/Y (*PackIpS2_1*) a partir da especificação do analisador de fluxo. Além disso, gerando-se diagramas MSC é possível analisar o comportamento do classificador de fluxo diante da chegada de pacotes IP no controlador do IpSwitch, a seleção dos fluxos a comutação e, a detecção de fluxos comutados inativos.

Nome	*Descrição*	*Data*	*Dia*	*Hora*	*Pacotes*
DEC-PKT-1	*Corporate Gateway*	08/03/95	Quarta	22:00	3.3 milhões
DEC-PKT-2	*Corporate Gateway*	09/03/95	Quinta	02:00	3.9 milhões
DEC-PKT-3	*Corporate Gateway*	09/03/95	Quinta	10:00	4.3 milhões
DEC-PKT-4	*Corporate Gateway*	09/03/95	Quinta	14:00	5.7 milhões
LBL-PKT-4	*Corporate Gateway*	21/01/94	Sexta	14:00	1.3 milhões
LBL-PKT-5	*Corporate Gateway*	28/01/94	Sexta	14:00	1.3 milhões
LBL-TCP-3	*Corporate Gateway*	20/01/94	Quinta	14:10	1.8 milhões

Tabela 3.6: Traces de Tráfego de Rede Internet

Os *traces* utilizados estão disponíveis no formato ASCII e apresentam 6 colunas : *TimeStamp* com precisão de microsegundos, *Host* Origem (renumerado), *Host* Destino (renumerado), Porta TCP Origem, Porta TCP Destino e, número de bytes transmitidos.

Durante as simulações só foram utilizadas as três primeiras colunas do *trace* que serviram de alimentação para a simulação da especificação do classificador de fluxo. As colunas de portas TCP origem e destino não foram utilizadas pois este trabalho utiliza nas simulações o tipo de fluxo 2 [14], que leva em consideração o endereço IP origem e o endereço IP destino como identificadores de fluxo sem utilizar as portas TCP, capítulo 2 seção IpSwitching. A coluna do número de bytes também não é utilizada, pois há uma preocupação somente com o número de pacotes transmitidos, e não com número de bytes transmitidos.

Através de um programa desenvolvido em C para este trabalho, os arquivos de trace de 6 colunas no formato ASCII, foram adaptados para servir de entrada para a simulação dos sistemas especificados no SDT *Simulator*. O programa em C, lê cada linha do *trace*, ignora as três últimas colunas e acrescenta os comandos do SDT *Simulator*, transformando as informações de um pacote IP em um sinal *IpPacket*. Cada linha do trace é transformada em comandos como o exemplo abaixo :

Output-To IpPacket 0.037912 3 4 <<Block FlowClass>> Analyzer
Next-Transition

O comando *Output-To IpPacket 0.37912 3 4 <<Block FlowClass>> Analyzer* significa que é enviado um sinal *IpPacket* com os parâmetros : 0.37912 representando o *Time-Stamp*, 3 representando o endereço IP origem e, 4 representando o endereço IP destino, para o processo *Analyzer* do bloco *FlowClass*. O comando *Next-Transition*, faz com que o sistema realize as transições devidas, após o recebimento de um sinal *IpPacket*.

3.7.1 Resultados da Simulação do Analisador de Fluxo (Versão 1.1)

Esta subseção apresenta os resultados obtidos em simulações realizadas com a especificação da versão 1.1, seção 3.4. O objetivo destas simulações é analisar as estatísticas gerais dos *traces* utilizados, como o número de pacotes transmitidos e o número de fluxos detectados.

A tabela 3.7 apresenta as estatísticas dos traces obtidas nas simulações onde a coluna *Pacotes* indica o número total de pacotes processados pelo simulador e, a coluna *Fluxos(2)* indica o número total de fluxos do tipo 2 detectados. A coluna *Med.Pac/Seg* indica a média de pacotes transmitidos por segundo e, a coluna *Med.Pac/Fluxo* indica a média de pacotes transmitidos por fluxo.

Trace	*Pacotes*	*Fluxos(2)*	*Med.Pac/Seg*	*Med.Pac/Fluxo*
DEC-PKT-1	147.718	1323	492	112
DEC-PKT-2	165.476	1208	552	137
DEC-PKT-3	258.182	2409	860	107
DEC-PKT-4	338.464	2533	1128	134
LBL-PKT-4	75.123	420	250	179
LBL-PKT-5	50.953	428	169	119
LBL-TCP-3	81.818	403	272	203

Tabela 3.7: Resultados das Simulações do Analisador de Fluxo (Versão 1.1)

O número de fluxos detectados depende das características de intensidade de tráfego. Tráfegos com diversas comunicações entre pares de endereços IP distintos e de curta duração, acarreta na detecção de muitos fluxos. Já tráfegos com poucas comunicações de curta duração apresentam um número menor de fluxos detectados, porém de maior duração e transmissão de um maior número de pacotes por fluxo.

É o que podemos observar por exemplo, analisando-se os dados estatísticos dos traces *DEC-PKT-1* e *DEC-PKT-2* na tabela 3.7. O trace *DEC-PKT-2* apresenta uma intensidade de transmissão de pacotes um pouco maior do que o trace *DEC-PKT-1*. O trace *DEC-PKT-2* transmitiu 17.758 pacotes TCP a mais do que o trace *DEC-PKT-1*. Entretanto, o trace *DEC-PKT-2* apresentou a detecção de 1208 fluxos e o trace *DEC-PKT-1*, apesar de transmitir uma quantidade menor de pacotes, apresentou a detecção de 1323 fluxos, ou seja, 115 fluxos a mais.

O mesmo acontece com os traces *LBL-PKT-4* e *LBL-PKT-5*. Durante os 5 minutos de simulação o trace *LBL-PKT-4* transmitiu 24.170 pacotes TCP a mais que o trace *LBL-PKT-5*. Entretanto, os dois traces apresentaram uma quantidade de fluxos detectados aproximadamente igual, uma média de 424 fluxos. Podemos concluir que o número de fluxos gerados depende exclusivamente do tipo de tráfego, isto é, se o tráfego apresenta muitos fluxos de curta duração ou, uma quantidade menor de fluxos de maior duração.

3.7.2 Resultados da Simulação do Classificador de Fluxo X/Y (Versão 2.1)

Esta subseção apresenta os resultados obtidos em simulações realizadas com a especificação descrita na seção 3.5, a Especificação do Classificador de Fluxo X/Y (Versão 2.1). Os resultados obtidos são mostrados através da tabela 3.8 apresentando dados estatísticos gerais sobre porcentagem de comutação de pacotes, fluxos detectados e fluxos comutados.

Trace	*Pacotes*	*PComutados*	*FCandidatos*	*FComutados*	%
DEC-PKT-1	147.718	115.292	1323	441	78,05
DEC-PKT-2	165.476	136.687	1208	487	82,60
DEC-PKT-3	258.182	200.465	2409	928	77,64
DEC-PKT-4	338.464	275.736	2533	866	81,47
LBL-PKT-4	75.123	62.671	420	181	83,42
LBL-PKT-5	50.953	39.669	428	176	77,85
LBL-TCP-3	81.818	70.155	403	162	85,75

Tabela 3.8: Estatísticas de Comutação de Pacotes e Comutação de Fluxos para X = 10 pacotes, Y = 20 segundos e Fluxos Comutados Inativos em 60 segundos

Os valores dos parâmetros do classificador de fluxo X/Y foram configurados para X = 10 pacotes e Y = 20 segundos. Além disso, a tabela de fluxos candidatos e fluxos comutados são verificadas de 60 em 60 segundos. Os fluxos comutados são considerados inativos se tornarem-se osciosos em 60 segundos, isto é, se permanecerem 60 segundos sem comutar pacotes.

Nos fluxos candidatos, durante a verificação da tabela de 60 em 60 segundos, se um fluxo não atingir os parâmetros X e Y, ele é reinicializado e analisado em um novo período de tempo como descrito na seção 3.5.

A tabela 3.8 apresenta os resultados das simulações referentes a comutação de pacotes e de fluxos. A primeira coluna, *Trace* apresenta o nome do *trace* utilizado na simulação. A coluna *Pacotes* apresenta o número total de pacotes TCP processados pelo simulador. A coluna *PComutados* apresenta o número total de pacotes que foram comutados. A coluna *FDetectados* apresenta o número total de fluxos do tipo 2 detectados como fluxos candidatos. A coluna *FComutados* apresenta o número de vezes que, fluxos do tipo 2, foram selecionados para a comutação. O mesmo fluxo pode ser selecionado para a comutação mais de uma vez durante a simulação. Este seria o caso em que um fluxo torna-se inativo e volta a transmitir pacotes posteriormente, sendo selecionado mais uma vez para a comutação. E finalmente a coluna % apresenta a porcentagem de pacotes comutados em relação ao total de pacotes processados pelo simulador.

Trace	*FReinic*	*FReinicP*	*FInativos*	*FInativosP*	*FAtivosNovamente*
DEC-PKT-1	3393	920	156	47	38
DEC-PKT-2	2875	753	163	57	32
DEC-PKT-3	4071	1319	245	113	69
DEC-PKT-4	4476	1502	290	118	58
LBL-PKT-4	939	252	60	16	13
LBL-PKT-5	1035	256	64	20	4
LBL-TCP-3	839	250	54	19	9

Tabela 3.9: Estatísticas dos Fluxos Detectados e Comutados

A tabela 3.9 apresenta os resultados referentes as características dos fluxos detectados e comutados. A coluna *Trace* apresenta o nome do trace utilizado. A coluna *FReinic* apresenta o número de vezes que foi realizada uma reinicialização de um fluxo candidato. Isso significa que, na verificação da tabela de fluxos candidatos que é realizada de 60 em 60 segundos, se um fluxo não atingir os parâmetros X e Y, ele é reinicializado para uma análise de um novo período de mais 60 segundos. Já a coluna *FReinicP* representa

o número de fluxos candidatos que foram reinicializados durante a última verificação da tabela de fluxos candidatos. A coluna *FInativos* apresenta o número total de vezes que se detectou um fluxo comutado inativo durante todo o tempo de simulação. Nas simulações realizadas considera-se um fluxo comutado inativo, se ele permanecer mais do que 60 segundos sem comutar pacotes. Já a coluna *FInativosP*, representa o número de fluxos comutados que foram considerados inativos durante a última verificação de tabela de fluxos comutados. A coluna *FAtivosNovamente* representa o número de vezes que fluxos comutados foram considerados inativos em um determinado instante da simulação, mas que voltaram a transmitir pacotes posteriormente.

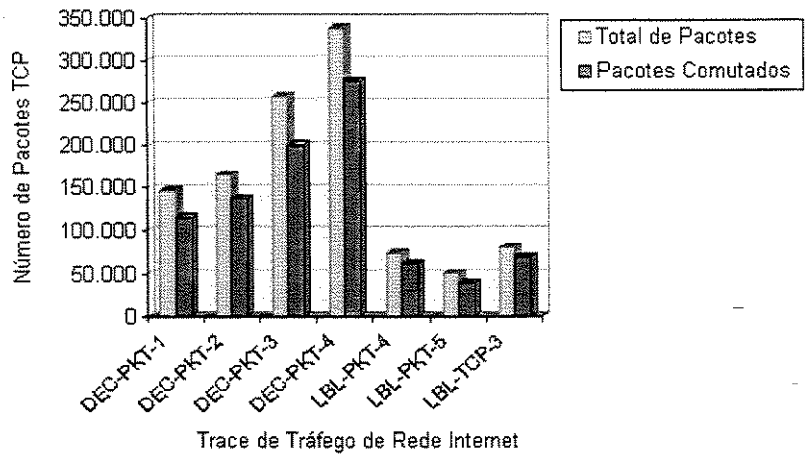

Figura 3.43: Pacotes Comutados em Relação ao Total de Pacotes Processados

O gráfico da figura 3.43 ilustra a relação do número total de pacotes processados e, o número total de pacotes comutados de cada *trace* durante todo o tempo de simulação. Pode-se observar que, o número de pacotes comutados em relação ao número total de pacotes processados, é na maioria dos *traces*, superior ou próximo a 80% utilizando-se os parâmetros X=10 e Y=20, tabela 3.8.

A determinação do tipo de classificador de fluxo e de seus parâmetros, influencia na eficiência do IpSwitching, isto é, na quantidade de pacotes comutados. A configuração dos parâmetros do classificador de fluxo deve ser adequada ao tipo de tráfego que se utilizará do IpSwitching.

Por exemplo, o trace *DEC-PKT-2* tem uma intensidade de transmissão de pacotes superior ao trace *DEC-PKT-1*, tabela 3.8, e teve uma detecção de fluxos detectados menor. Entretanto, o número de fluxos comutados como pode ser observado na tabela 3.8, foi

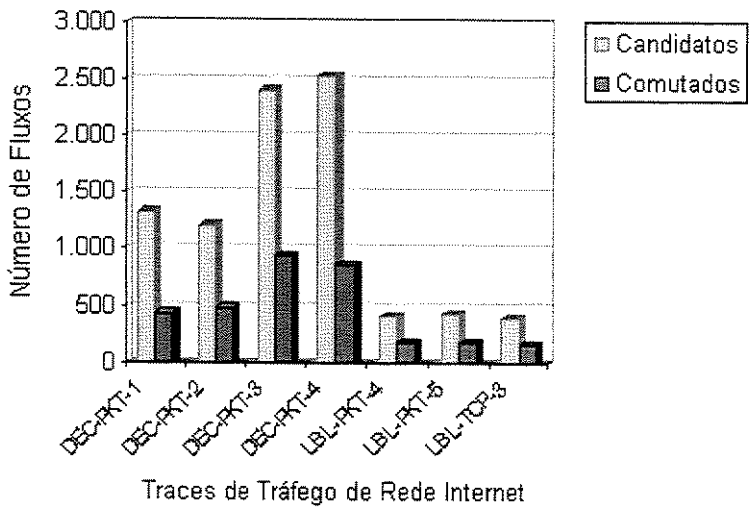

Figura 3.44: Número de Fluxos Detectados e Quantidade de Fluxos Comutados

maior do que o o número de fluxos comutados do trace *DEC-PKT-1*. Podemos concluir que a escolha dos parâmetros X=10, Y=20 adotados na simulação foi mais adequado para o tipo de trace *DEC-PKT-2* do que para o trace *DEC-PKT-1*. A escolha dos parâmetros é muito particular a cada tipo de tráfego que depende mais uma vez, da quantidade de pacotes transmitidos por cada fluxo e da duração média destes fluxos. A subseção 3.7.3 aborda esta questão com mais detalhes.

O gráfico da figura 3.44 ilustra a relação do número de fluxos candidatos e o número de fluxos comutados. É importante lembrar que um mesmo fluxo pode ser selecionado mais de uma vez para comutação, no caso de tornar-se inativo, voltar a transmitir pacotes, e ser selecionado novamente de acordo com os parâmetros X e Y do classificador.

3.7.3 Resultados da Simulação da Variação dos Parâmetros X e Y do Classificador de Fluxo

Esta subseção apresenta resultados obtidos de simulações com a versão de especificação do Classificador de Fluxo 2.1 e, com o arquivo de trace *LBL-PKT-4*. Escolheu-se o arquivo *LBL-PKT-4* por não ser um arquivo de grande intensidade de tráfego como o *DEC-PKT-3* ou *DEC-PKT-4*. Não se utilizou um arquivo de maior intensidade de tráfego, somente pelo motivo de requerer tempos de simulação maiores.

O objetivo desta seção é analisar o comportamento do Classificador de Fluxo e da eficiência do IpSwitching diante da variação dos parâmetros X e Y.

A tabela 3.10 apresenta os dados obtidos nas simulações. As colunas X e Y representam os valores dos parâmetros do classificador de fluxo utilizados nas simulações. A coluna *PComutados* representa o número total de pacotes comutados durante a simulação.

X	Y	*PComutados*	*FComutados*	*FInativos*	*FAtivosNovamente*	%
5	1	72.227	340	179	19	96,14
10	5	68.576	233	102	16	91,28
15	10	63.745	157	50	10	84,85
25	20	61.394	146	45	10	81,72
30	25	59.383	133	40	9	79,04
35	30	57.182	119	34	9	76,12
40	35	52.992	106	27	8	70,54
45	40	49.132	93	21	5	65,40

Tabela 3.10: Variação dos Parâmetros X e Y, Total de Pacotes Processados = 75.123

A coluna *FInativos* representa o número de vezes que se detectou um fluxo comutado inativo. A coluna *FAtivosNovamente* representa o número de fluxos comutados que foram considerados inativos mas que voltaram a transmitir pacotes. E finalmente a coluna % representa a porcentagem de pacotes comutados em relação ao número total de pacotes processados pelo simulador.

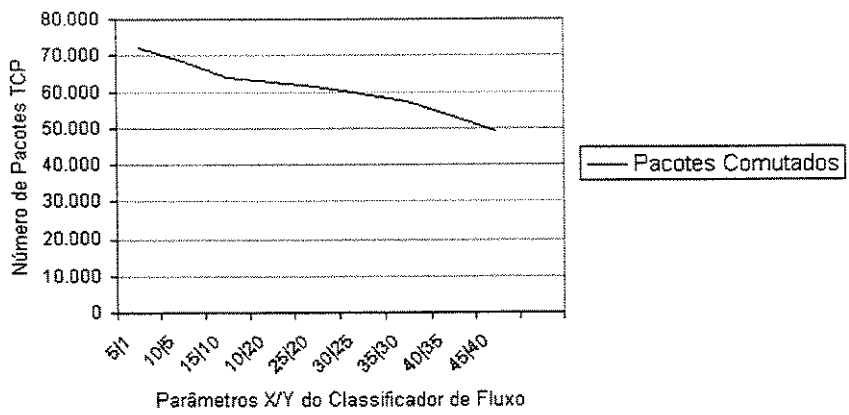

Figura 3.45: Pacotes Comutados em Relação à Variação dos Parâmetros X e Y do Classificador de Fluxo

Na simulação do trace *LBL-PKT-4* (apenas 5 minutos), foi processado um total de 75.123 pacotes TCP. O gráfico da figura 3.45 ilustra o número de pacotes TCP comutados diante da variação dos parâmetros X e Y.

Durante as simulações os valores X e Y foram aumentados gradativamente. Observa-se que o número de pacotes comutados diminui com o aumento dos valores X e Y. Para os valores X=5/Y=1 a performance é muito próxima de 100 %. Além disso foram detectados 340 seleções de fluxos a comutação dentro de 420 fluxos detectados. Entretanto, detectou-se 179 vezes a existência de fluxos inativos.

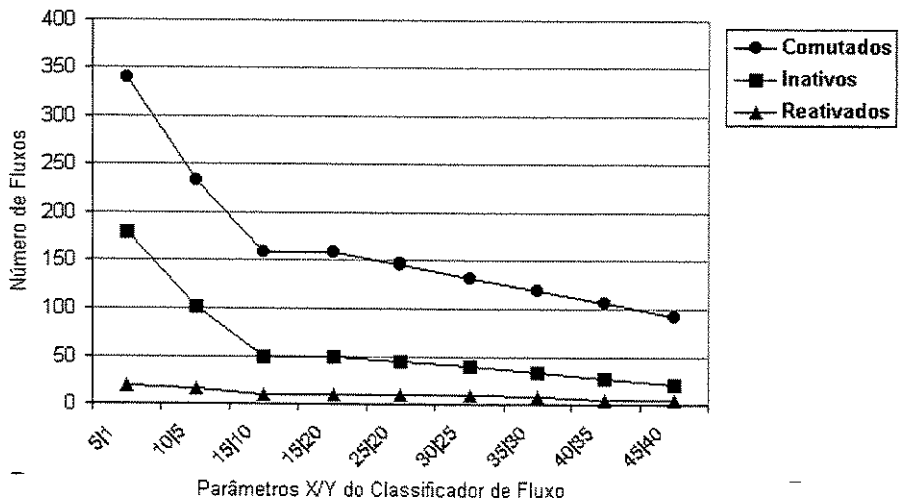

Figura 3.46: Fluxos Comutados, Inativos e Reativados em Relação a Variação dos Parâmetros X e Y do Classificador de Fluxo

Isto significa que, muitos dos fluxos selecionados para a comutação apresentavam curta duração. É desejável selecionar para a comutação todos ou a maioria dos fluxos. Por outro lado, isso requer a alocação de uma quantidade grande de VCs, sendo que em alguns equipamentos de comutação, o número de VCs pode ser limitado.

O gráfico da figura 3.46 ilustra o número de fluxos comutados, o número de vezes de detecção de fluxos inativos e, o número de fluxos que tornaram-se inativos e voltaram a transmitir pacotes.

Observa-se que tanto o número de fluxos candidatos o número de fluxos inativos e o número de fluxos que tornaram-se ativos novamente, decrescem com o aumento dos parâmetros X e Y.

Conclui-se que, é sempre desejável uma porcentagem de comutação maior possível. Entretanto, a configuração dos parâmetros X e Y dependem da disponibilidade de VCs do equipamento utilizado. Valores de X e Y que permitam uma porcentagem de comutação maior, geram mais fluxos comutados e vão requerer portanto, uma quantidade maior de

VCs disponíveis.

3.7.4 Resultados da Simulação da Variação do Parâmetro de Fluxos Inativos

Esta subseção procura estudar a influência da variação do parâmetro de fluxo inativo na quantidade de pacotes comutados, na quantidade de fluxos comutados, inativos e ativos novamente.

Neste trabalho, considera-se como parâmetro de fluxo inativo, o parâmetro que estabelece quando um fluxo será considerado inativo. Se este parâmetro for configurado para o valor 60 por exemplo, um fluxo selecionado para a comutação será considerado inativo se em 60 segundos ele permanecer sem comutar qualquer pacote.

As simulações foram realizadas variando-se o parâmetro de fluxo inativo de 10 a 60. Todas as simulações foram realizadas utilizando-se como entrada o arquivo de trace *LBL-PKT-4*. Os parâmetros do classificador de fluxo utilizados nas simulações foram X=15 e Y=10. As tabelas de fluxos candidatos e de fluxos comutados são verificadas de 30 em 30 segundos.

Inat	*Comutados*	*FInativos*	*FInativosP*	*FAt.Nov*	*FComutados*	%
10	63.464	161	16	101	235	84.48
20	64.447	106	12	49	190	85.79
30	64.706	84	8	31	177	86.13
40	64.889	70	9	21	169	86.38
50	64.987	60	11	13	163	86.51
60	65.025	56	9	11	161	86.56

Tabela 3.11: Variação do Parâmetro de Fluxo Inativo, X = 15 pacotes e Y = 10 segundos, Total de Pacotes Processados = 75.123

A tabela 3.11 apresenta os dados obtidos nas simulações. A coluna *Inat* apresenta o valor do parâmetro de fluxo inativo. A coluna *Comutados* apresenta o número total de pacotes comutados. A coluna *FInativos* apresenta o número de vezes que fluxos inativos foram detectados. A coluna *FInativosP* apresenta o número de fluxos que foram considerados inativos durante a última verificação de tabela de fluxos comutados. A coluna *FAt.Nov* apresenta o número de fluxos comutados que tornaram-se inativos, mas que voltaram a transmitir pacotes.

A coluna *FComutados* apresenta o número de vezes que fluxos foram selecionados para a comutação, e finalmente a coluna %, apresenta a porcentagem de pacotes comutados

em relação ao total de pacotes processados pelo simulador.

O número total de pacotes processados pelo simulador com o arquivo *LBL-PKT-4* foi de 75.123 e, o número total de fluxos detectados foi de 420.

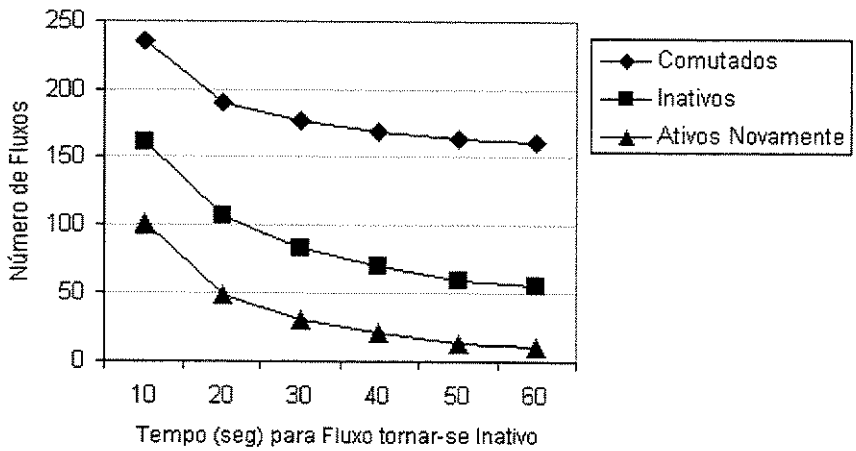

Figura 3.47: Estatísticas dos Fluxos Comutados diante da Variação do Parâmetro de Fluxo Inativo

O gráfico da figura 3.47 apresenta as estatísticas dos fluxos detectados e comutados, em relação à variação do parâmetro de fluxo inativo. Observa-se que com a variação do parâmetro de fluxo inativo de 10 a 60 segundos, há uma diminuição na quantidade de fluxos comutados, fluxos inativos e fluxos ativos novamente.

Quando foi utilizado o valor do parâmetro de fluxo inativo igual a 10, detectou-se 235 seleções de fluxos à comutação. Entretanto, detectou-se 161 ocorrências de fluxos inativos. Além disso, 101 ocorrências de fluxos comutados que foram considerados inativos e, que voltaram a transmitir pacotes. Conclui-se que, um valor pequeno para o parâmetro de fluxo inativo não é apropriado. A alocação de VCs disponíveis, a remoção dos fluxos comutados da tabela quando tornam-se ativos, e novamente a alocação de VCs quando estes fluxos voltam a transmitir pacotes, gera um atraso no processamento.

Já quando o valor de parâmetro inativo foi de 60 segundos, observa-se uma detecção bem menor de fluxos inativos, assim como uma detecção menor de fluxos que foram considerados inativos e voltaram a transmitir pacotes.

O gráfico da figura 3.48, apresenta a porcentagem de comutação de pacotes em relação à variação do parâmetro de fluxo inativo. Observa-se que, variando-se o parâmetro de fluxo inativo de 10 a 60 segundos, houve um pequeno incremento na quantidade de pacotes comutados. Isto ocorre porque, com valores do parâmetro de fluxo inativo pequenos,

poucos segundos após um fluxo ter sido selecionado para comutação e começar a comutar pacotes, este mesmo fluxo já é considerado inativo. Desta forma, o fluxo volta a transmitir pacotes em nível de camada 3 e, leva um período transmitindo estes pacotes até que seja selecionado para a comutação novamente.

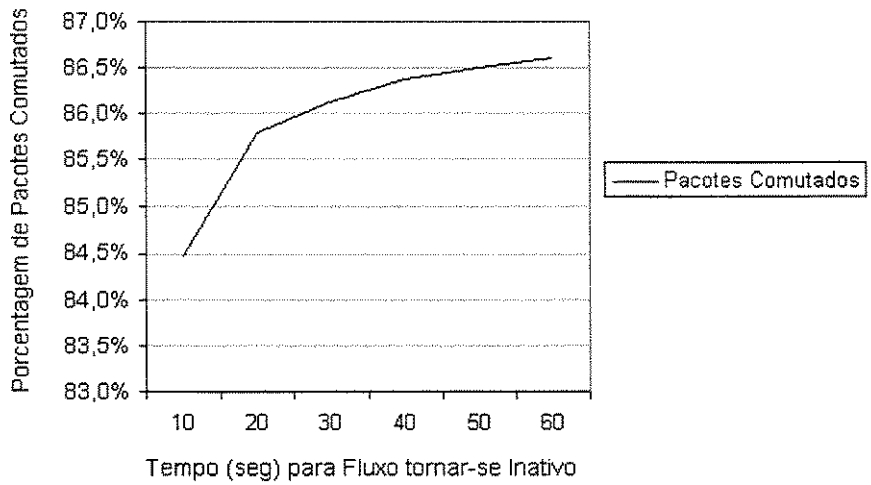

Figura 3.48: Porcentagem de Comutação em Relação à Variação do Parâmetro de Fluxo Inativo

Se o fluxo não tivesse sido considerado inativo em tão poucos segundos, os pacotes que foram transmitidos no nível de camada 3, teriam sido comutados. Por este motivo, quando utiliza-se valores do parâmetro de fluxo inativo maiores, a quantidade de pacotes comutados é maior.

Se o fluxo não tivesse sido considerado inativo em tão poucos segundos, os pacotes que foram transmitidos no nível de camada 3, teriam sido comutados. Por este motivo, quando utiliza-se valores do parâmetro de fluxo inativo maiores, a quantidade de pacotes comutados é maior.

3.8 Considerações sobre o Capítulo 3

O classificador de fluxo do tipo X/Y foi escolhido para as especificações deste capítulo porque, através da análise da variação dos parâmetros X, Y e do parâmetro de fluxo inativo, além da análise de incidência de tráfego, pode-se controlar a taxa de fluxos transmitidos pelo roteamento *hp-by-hop* do IP padrão e, a taxa de fluxos comutados diretamente no *hardware* de camada 2.

Nas simulações dos diferentes *traces* apresentadas na seção 3.7.2, tabela 3.8, com a utilização dos parâmetros X = 10 pacotes, Y = 20 segundos e fluxo do tipo 2 pelo menos 77,64% dos pacotes foram comutados.

Nas simulações apresentadas em [23], pelo menos 75% dos pacotes foram comutados utilizando-se os mesmos *traces* e os mesmos parâmetros do classificador de fluxo. Segundo [23], este resultado não satisfaz a proposta dos inventores do IpSwitching em [26] que declaram que, quando é utilizado o tipo de fluxo 2, pelo menos 80% dos pacotes serão comutados. Os autores do IpSwitching consideram uma comutação de 80

Apesar dos resulados deste trabalho serem muito próximos de [23] é conveniente lembrar que, nas simulações do capítulo 3 foi utilizado apenas 5 minutos dos *traces* de tráfego IP e, foram analisados somente os pacotes TCP. As simulações realizadas em [23] utilizam uma hora de *trace* e portanto, os resultados devem ser mais precisos.

Apesar do classificador de fluxo ser parte integrante do controlador do IpSwitch e ser de grande importância para o desempenho do IpSwitching, seu comportamento funcional não é especificado em [14].

Neste capítulo, através do uso da linguagem SDL foi possível criar uma especificação formal do classificador de fluxo definindo o seu comportamento funcional. Através do uso do *Simulator* do SDT e dos *traces* de tráfego IP, foi possível além da obtenção de dados estatísticos sobre o desempenho do IpSwitching, gerar diagramas MSC que expressam o comportamento do classificador diante de diversas situações.

Capítulo 4

Especificação e Simulação da Máquina de Estado e dos Procedimentos do LDP

4.1 Introdução

No capítulo 3 foi apresentada a especificação do classificador de fluxo do tipo X/Y, que é parte integrante de um IpSwitch. A escolha do classificador de fluxo, assim como a configuração dos parâmetros do classificador, podem exercer grande influência na performance do protocolo IpSwitching. Desta forma, o objetivo da especificação do capítulo 3, era analisar o desempenho do protocolo IpSwitching diante da variação dos parâmetros do classificador de fluxo X/Y.

O MPLS (*MultiProtocol Label Switching*) é um protocolo, assim como o IpSwitching, que utiliza a comutação de camada 2 para transportar pacotes de camada 3. O MPLS foi criado pelo MPLS *Working Group* com o objetivo de padronizar uma tecnologia que combina a técnica de *label-swapping* com o roteamento de camada 3 (seção 2.5, cap. 2). O MPLS utiliza um protocolo para distribuição, manutenção e remoção dos mapeamentos de *labels* à FECs (*Forwarding Equivalence Classes*), chamado *Label Distribution Protocol* (LDP) e que é objeto de estudo deste capítulo.

Este capítulo apresenta na subseção 4.2 uma expansão dos diagramas da máquina de estado do LDP para LSRs (*Label Switching Router*) ATM sem capacidade de realizar *VC-merge*, baseado nos diagramas apresentados em [24]. Na subseção 4.3 é apresentado a combinação da especificação formal em SDL dos procedimentos de distribuição de *label* baseado em [21], com a máquina de estado do LDP da seção 4.2. A subseção 4.4 apresenta os resultados obtidos nas simulações utilizando o SDT (*SDL Design Tool*) da especificação da seção 4.3. Finalmente a subseção 4.5 apresenta algumas considerações sobre o cap. 4.

As especificações e as simulações apresentadas neste capítulo, diferentemente do capítulo 3, têm o objetivo de realizar uma análise comportamental do LDP, e não de coletar dados estatísticos sobre o desempenho do protocolo.

O internet-draft [24] apresenta uma proposta de máquina de estado para LSRs ATM

sem capacidade de realizar *VC-merge* e, uma segunda máquina de estado para LSRs ATM capazes de realizar *VC-merge*. Como os comutadores ATM podem apresentar limitações no número de espaço na tabela de *cross-connect*, se não forem capazes de realizar *VC-merge*, é necessário que estes LSRs ATM se utilizem do modo de distribuição de label *DownStream-On-Demand* e, modo de retenção de label *Conservativo*, seção 2.5 do cap. 2.

Portanto, a máquina de estado proposta em [24], para LSRs ATM sem capacidade de realizar *VC-merge*, restringe-se aos procedimentos de distribuição de label *DownStream-On-Demand* Ordenado ou, *DownStream-On-Demand* Independente, ambos procedimentos com retenção de label *Conservativo*.

As máquinas de estado LDP propostas em [24], são mostradas através de um diagrama que representa a transição entre estados e os possíveis eventos e, através de uma explicação textual . A explicação textual consiste em, dado um estado, após a ocorrência dos possívies eventos definidos, a ação que o LSR toma e, a transição para um novo estado ou, a permanência no mesmo estado, caso não haja transição.

Este trabalho propõe na seção 4.2 (*Proposta de Diagramas para a Máquina de Estado do LDP*), uma expansão dos diagramas de máquina de estado do LDP apresentado em [24], para LSRs incapazes de realizar *VC-merge*. Esta expansão consiste em desenvolver diagramas que representem as transições de estados da máquina do LDP não só para LSRs ATM. Isto significa que a máquina de estado proposta neste trabalho apresenta diagramas que dizem respeito a LSRs que possam estar operando em qualquer um dos modos de distribuição de label, *DownStream* ou *DownStream-On-Demand*, em qualquer um dos modos de controle LSP, *Independente* ou *Ordenado* e, em qualquer um dos modos de retenção de label, *Conservativo* ou *Liberal*, mas para LSRs não *merge*.

Além da definição das transições de estados do LDP, os diagramas propostos na seção 4.2, apresentam uma breve descrição das principais ações a serem tomadas pelo LSR, diante da ocorrência de um evento que cause mudança de estado. A descrição textual dos diagramas apresentadas nas seções 4.2.1, 4.2.2, 4.2.3 e 4.2.4 diz respeito à descrição somente dos eventos que causam mudanças de estados.

O internet-draft [21] apresenta o comportamento de um LSR operando em qualquer um dos procedimentos de distribuição de *labels* do LDP, diante do recebimento de mensagens do LDP, ou da ocorrência de eventos como a detecção de um novo *Next-Hop* para uma FEC (*Forwarding Equivalence Class*), subseção 2.5.3 capítulo 2. A descrição do comportamento do LSR é apresentado de forma textual e é dividida em três partes :

1. *Descreve brevemente a resposta do LSR ao evento occorido*,
2. *Uma lista dos elementos referenciados pelo algoritmo da terceira parte* e,
3. *Um algoritmo para a resposta do LSR ao evento.*

O algoritmo definido na terceira parte, não apresenta a estrutra de dados que deve ser usada para armazenar as informações sobre a atividade do protocolo, mas especifica quais informações devem ser armazenadas, e assume que elas possam ser recuperadas quando necessário.

A especificação apresentada na seção 4.3, baseou-se nos algoritmos descritos em [21] e utilizou a estrutura de bloco de controle LSP definido em [24], como estrutura de dados para armazenar as informações sobre cada um dos LSPs (*Label Switched Path*). Com esta estrutura, é possível gerar-se uma máquina de estado LDP para cada LSP criado (que não é descrita em [21]).

O objetivo da especificação é combinar as informações textuais presentes nos dois documentos referenciados para construir uma especificação formal do protocolo LDP, descrevendo o comportamento do LSR e a máquina de estado do LDP. Além da tradução das informações textuais para a especificação formal, foram definidas estruturas de dados para armazenar as informações necessárias para a operação do LDP. A especificação diz respeito a um LSR capaz de operar em qualquer um dos procedimentos de distribuição de label, controle LSP e modo de retenção de label e que, pode receber e enviar mensagens LDP aos seus *LDP Peers DownStream* e, aos seus *LDP Peers UpStream.*

A simulação do sistema especificado neste capítulo, seção 4.4, foi realizada utilizando-se o *Simulator* do SDT. Os resultados são apresentados através de diagramas MSC, que são gerados com o envio de uma sequência de sinais do ambiente externo ao sistema simulado. As simulações procuram verificar se o comportamento do LSR diante de diferentes configurações, ocorreu de acordo com os procedimentos descritos em [21]. Além disso, verifica-se como se formam as máquinas de estado do LDP, que vão sendo geradas de acordo com o estabelecimento de LSPs.

4.2 Proposta de Diagramas para a Máquina de Estado do LDP

Esta seção apresenta uma proposta de máquina de estado do LDP para LSRs que possam operar em qualquer um dos procedimentos de distribuição de label, *DownStream* ou *DownStream-On-Demand*, qualquer um dos modos de controle LSP, *Independente* ou *Ordenado* e, qualquer um dos modos de retenção de label, *Conservativo* ou *Liberal.*

A proposta da máquina de estado do LDP é representada através de quatro diagramas, que são apresentados e descritos textualmente nas subseções 4.2.1, 4.2.2, 4.2.3 e 4.2.4. Os quatro diagramas dizem respeito a LSRs que operam sem capacidade de realizar *VC-merge.*

O objetivo de se dividir a máquina de estado do LDP em quatro partes, é de criar diagramas mais específicos em relação à configuração do LSR, e não um diagrama genérico como o apresentado em [24]. Dividindo-se os diagramas de acordo com o modo de controle LSP e o tipo de distribuição de label, pode-se entender mais facilmente o comportamento do LSR diante da ocorrência dos eventos e de acordo com a configuração do LSR.

O comportamento das máquinas de estado desta seção é especificado na seção 4.3. A máquina de estado do LDP está embutida na especificação através da utilização da estrutura de controle de LSP, capítulo 2 (2.5.13). Para cada LSP a ser estabelecido é criado um registro de estrutura de controle LSP que armazena as informações sobre o LSP. O estado do LSP, isto é, o estado da máquina do LDP é armazenado em um dos

Estado	Descrição
IDLE	Estado inicial do LSP, quando o controle de bloco LSP é criado.
RESPONSE AWAITED	Este estado significa que, o LSR recebeu um *Request* de um LSR *UpStream*, processou o *Request*, enviou um novo *Request* em direção ao LSR *DownStream* e, está esperando por uma mensagem de *Mapping* do LSR *DownStream*.
ESTABLISHED	Este estado significa que o LSR recebeu um *Mapping* do LSR *DownStream* e, o LSP está estabelecido e operacional (na visão do LSR em questão). Para o LSP estar neste estado, deve haver um label *UpStream* e um label *DownStream* alocados.
NEW NH RESPONSE AWAITED	Este é o estado onde o LSR está no meio do estabelecimento de um LSP para um novo *Next-Hop*. O LSR enviou uma mensagem de *Request* para o novo *Next-Hop* e, está esperando que o novo *Next-Hop* envie uma mensagem de *Mapping*.

Tabela 4.1: Descrição dos Estados da Máquina de Estado do LDP

campos da estrutura de dados, seção 4.3. Por isso, diz-se que a máquina de estado do LDP está embutida na especificação dos procedimentos de distribuição de *labels*.

Os diagramas de máquina de estados do LDP apresentados nas figuras 4.1, 4.2, 4.3 e 4.4 são estruturados da seguinte forma :

- o símbolo quadrado de cantos arredondados representa os possíveis estados da máquina LDP .

- os arcos com setas entre os estados indicam a direção das transições.

- os losângos com linhas tracejadas representam uma condição de verificação. Se a condição for verdadeira, segue-se o caminho especificado pela palavra *Verdadeiro*, caso contrário, segue-se o caminho indicado pela palavra *Falso*.

- o texto apresentado em letras em itálico, representam as ações mais importantes que serão tomadas pelo LSR, antes que um LSP mude de estado.

- os eventos são representados pelas mensagens LDP escritas com letras em itálico e negrito.

A tabela 4.1 apresenta os possíveis estados da máquina de estado do LDP e, suas descrições de acordo com a definição apresentada em [24].

O estado *NEW NH RETRY* definido em [24], é o estado onde o LSR espera por um temporizador expirar e então, tenta estabelecer um LSP através de um novo *Next-Hop*. Neste estado não é tomada nenhuma ação, e este estado deve provavelmente representar

o estado onde o LSR realiza os procedimentos de prevenção de *loop*. Como a proposta da máquina de estado do LDP deste trabalho não abrange prevenção de *loop*, este estado foi excluído dos diagramas.

A subseção 4.2.1 apresenta o diagrama da máquina de estado LDP para LSRs operando no modo de controle LSP *Independente* e, a seção 4.2.2, o diagrama da máquina de estado LDP para LSRs operando no modo de controle LSP *Ordenado*.

As subseções 4.2.3 e 4.2.4, apresentam os diagramas de máquina de estado LDP, para LSRs operando no modo de distribuição de label *DownStream* e, *DownStream-On-Demand* respectivamente.

4.2.1 Diagrama da Máquina de Estado do LDP para LSRs operando no Modo de Controle LSP Independente

Esta subseção descreve a proposta de um diagrama de máquina de estado LDP, figura 4.1, para LSRs não *merge* e que, estejam operando no modo de controle LSP *Independente*. Neste diagrama, não é levado em consideração o tipo de distribuição de label, *DownStream* ou *DownStream-On-Demand* e, o tipo de retenção de label, *Conservativo* ou *Liberal*. Isto significa que, o LSR pode estar operando em qualquer um dos modos de configuração , mas que esteja configurado para operar no modo de controle LSP *Independente*.

Na figura 4.1, pode-se observar que o LSP encontra-se inicialmente no estado *Idle*. Ao ocorrer o evento de recebimento de uma mensagem de requisição de label de um LDP *Peer UpStream*, o LSR analisa a condição : *O LSR atua como Nó De Saída Da Rede para a FEC?* ou, *O LSR já recebeu e manteve um label mapping do Next-Hop para a FEC?*. Se a condição for verdadeira o LSR aloca um label *upstream* disponível, associa o label *upstream* recém alocado ao label *downstream* previamente recebido de um mapeamento do próximo *hop* e, envia uma mensagem de mapeamento ao *Peer Upstream* requisitante com o label alocado para a FEC. Após tomar estas ações, o estado do LSP para a FEC passa de *Idle* para *Established*.

Caso a condição não seja satisfeita, o LSR não possui um label *downstream* para estabelecer o LSP. Desta forma, o LSR, como está operando no modo independente, aloca um label *upstream* mesmo que não tenha recebido um mapeamento do próximo *hop* para a FEC. Este tipo de procedimento pode fazer com que, pacotes sem label sejam propagados do LSR para o *Peer DownStream*, se o *Peer UpStream* começar a transmitir pacotes antes que o LSR receba um label *downstream*.

O LSR, envia então uma mensagem de mapeamento ao *Peer UpStream* requisitante e, propaga a mensagem de requisição de label para o *Next-Hop DownStream*. O estado do LSP muda de *Idle* para *Response Awaited*.

Se o LSP estiver no estado *Response Awaited* e, receber uma mensagem de mapeamento do próximo *hop* para a FEC, o LSR associa o label *downstream* recebido na mensagem, ao label *upstream* previamente alocado. Não é necessário propagar a mensagem de mapeamento, já que o label *upstream* foi alocado e propagado quando o *Peer Upstream* enviou a mensagem de requisição de label. Neste caso, o LSP muda do estado *Response Awaited* para *Established*.

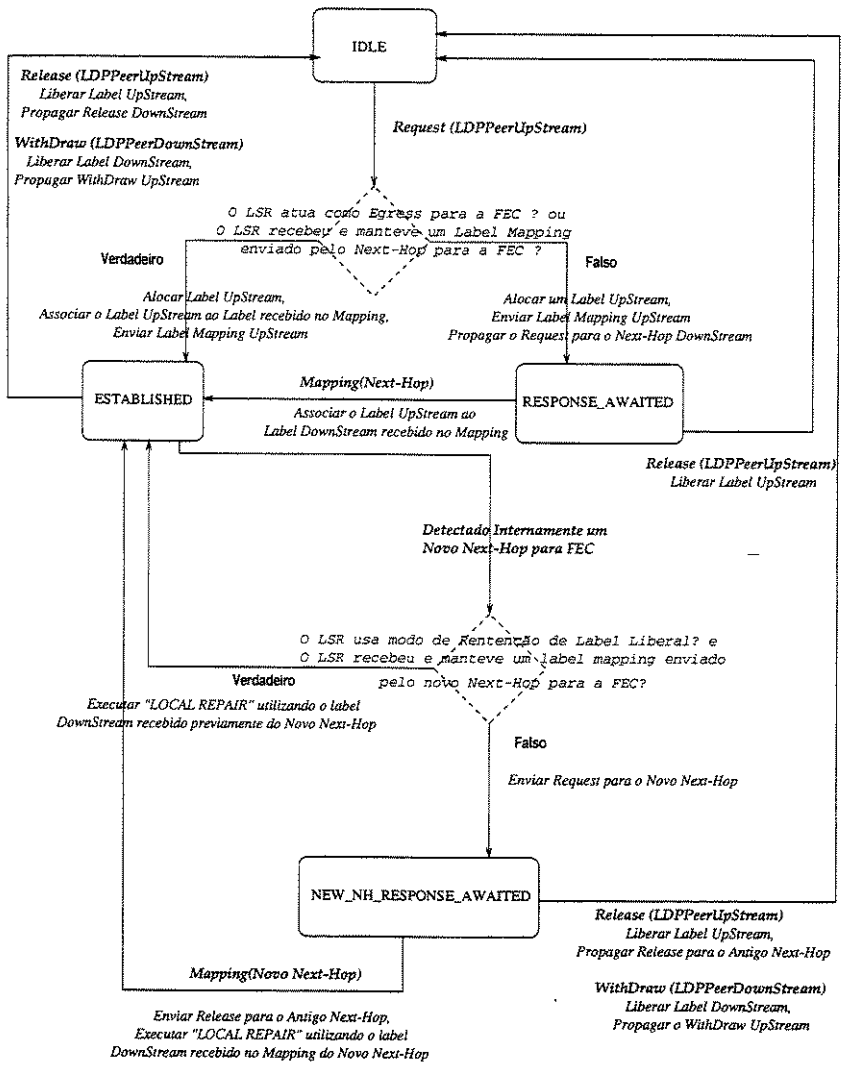

Figura 4.1: Diagrama da Máquina de Estado do LDP para LSRs operando no Modo de Controle LSP Independente

Se o LSP estiver no estado *Response Awaited* e receber uma mensagem de *Release* de um LDP *Peer UpStream*, o LSR libera o label *upstream* para a FEC e, o LSP muda do estado *Response Awaited* para *Idle*.

Se o LSP estiver no estado *Established* e, o LSR receber uma mensagem de *Release* de um LDP *Peer Upstream*, o LSR libera o label *upstream* para a FEC e, propaga a mensagem de *Release* para o LDP *Peer DownStream* da FEC.

Se o LSP estiver no estado *Established* e, o LSR receber uma mensagem de *Withdraw* de um LDP *Peer DownStream*, o LSR libera o label *downstream*, envia uma mensagem de *Release* para o *Peer DownStream* que enviou o *Withdraw*, confirmando a liberação do label *downstream* e, propaga a mensagem de *Withdraw* para os LDP *Peers UpStream* à FEC.

Se o LSP estiver no estado *Established* e, o LSR detectar internamente uma mudança de próximo *hop* para a FEC, ele analisa a condição : *O LSR usa modo de retenção de label Liberal?* e, *O LSR recebeu e manteve um label mapping enviado pelo novo Next-Hop?*

Desta forma, se as duas condições forem verdadeiras, o LSR comporta-se como se tivesse acabado de receber uma mensagem de mapeamento do novo *Next-Hop* e realiza a troca de próximo *hop* executando o procedimento de *LOCAL REPAIR*. O LSP continua no estado *Established*.

Se o LSR estiver operando no modo de retenção de label conservativo, ele provavelmente não tem armazenado informações sobre um mapeamento enviado do próximo *hop* novo, já que no modo conservativo, somente os *Mappings* referentes a FECs reconhecidas pelo LSR são armazenados. O LSR, se conservativo,ou liberal, mas que não tenha recebido um mapeamento do próximo *hop*, envia uma mensagem de requisição de label para o próximo *hop* novo. O LSP passa para o estado *New NH Response Awaited*.

Estando no estado *New NH Response Awaited*, ao receber uma mensagem de mapeamento do próximo *hop* novo, o LSR realiza o procedimento de *LOCAL REPAIR* e, o LSP vai para o estado *Established*. O label *downstream* recebido no mapeamento do próximo *hop* novo é associado ao label *upstream*. O LSR envia também, uma mensagem de *Release* para o *Antigo Next-Hop*.

Estando no estado *New NH Response Awaited*, ao receber uma mensagem de *Release* de um LDP *Peer UpStream*, o LSR libera o label *upstream* para a FEC e, propaga o *Release* para o antigo próximo *hop*. O LSP passa do estado *New NH Response Awaited* para *Idle*

Já ao receber uma mensagem de *Withdraw*, o LSR libera o label *downstream* e propaga a mensagem de *Withdraw* para o LDP *Peer UpStream*. O LSP passa do estado *New NH Response Awaited* para *Idle*.

4.2.2 Diagrama da Máquina de Estado LDP para LSRs operando no Modo de Controle LSP Ordenado

Esta seção apresenta a descrição, figura 4.2, da máquina de estado LDP para LSRs operando no modo de controle LSP Ordenado, isto é, *DownStream* Ordenado ou *DownStream-On-Demand* Ordenado.

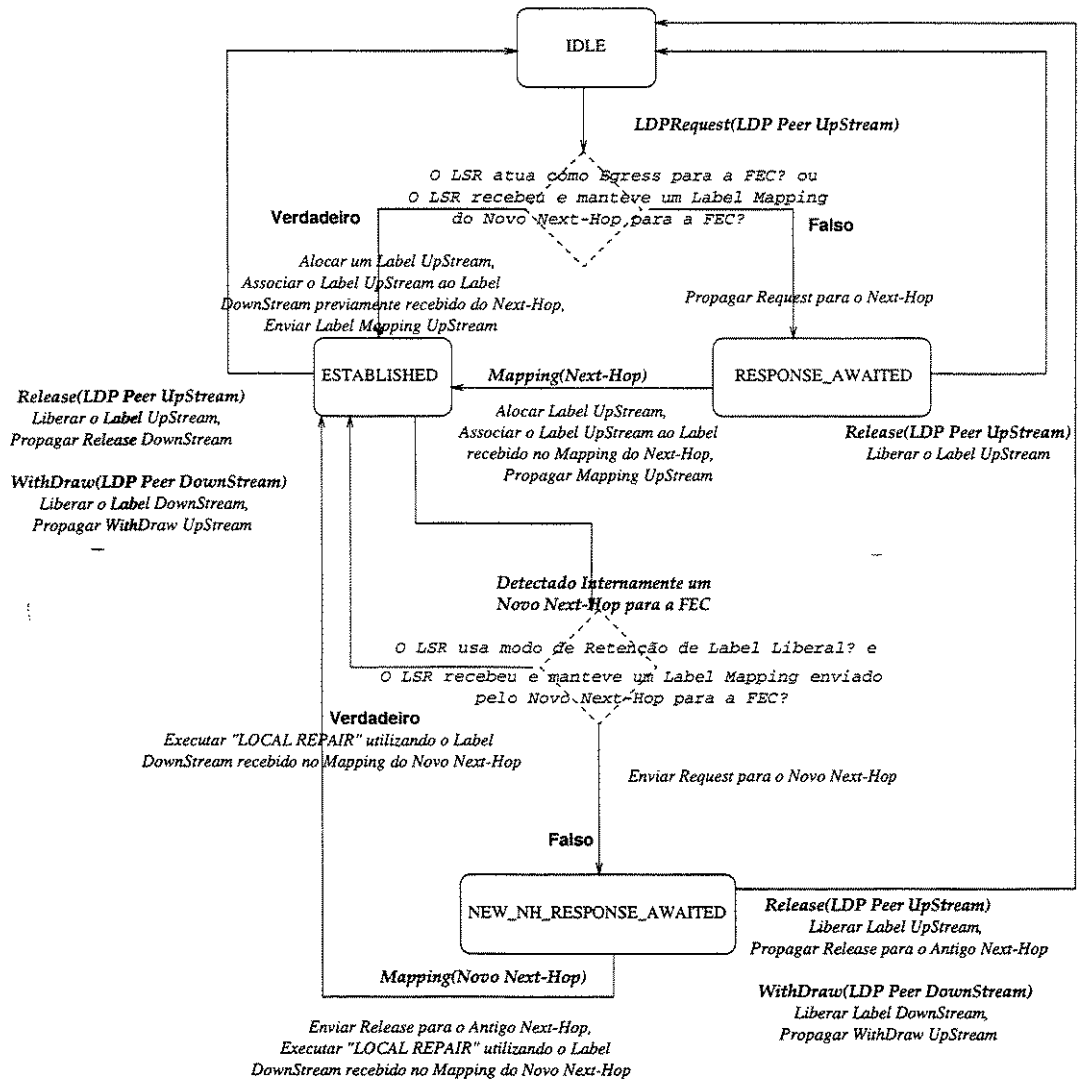

Figura 4.2: Diagrama da Máquina de Estado do LDP para LSRs operando no Modo de Controle LSP Ordenado

De acordo com a figura 4.2, o LSP para uma determinada FEC, encontra-se inicialmente no estado *Idle*. Se o LSR receber uma mensagem de requisição de label de um *Peer UpStream*, o LSR analisa a condição : *O LSR é Nó De Saída Da Rede para a FEC?* ou, *O LSR recebeu e manteve um label mapping do Next-Hop para a FEC?*

Se uma das duas condições for verdadeira, o LSR aloca um label *upstream*, conecta o label *upstream* alocado ao label *downstream* previamente recebido do próximo *hop* para a FEC e, envia um *Mapping UpStream*. O LSP muda do estado *Idle* para o estado *Established*.

Entretanto, se o LSR não for nó de saída da rede e, não tiver recebido um mapeamento do próximo *hop*, ele não aloca um label *upstream*. Ele somente propaga a mensagem de requisição de label para o *Next-Hop DownStream* e, permanece no estado *Response Awaited* até que receba um mapeamento do próximo *hop*.

Estando o LSP no estado *Response Awaited* e, recebendo um mapeamento do próximo *hop*, o LSR então, aloca um label *upstream*, associa o label *upstream* alocado ao label *downstream* recém recebido e, envia o *Mapping UpStream*. O LSP muda do estado *Response Awaited* para o estado *Established*.

Se o LSP estiver no estado *Response Awaited* e receber uma mensagem de *Release* de um LDP *Peer UpStream*, o LSR libera o label *upstream* para a FEC e, o LSP muda do estado *Response Awaited* para *Idle*.

Se o LSP estiver no estado *Established* e, o LSR receber uma mensagem de *Release* de um LDP *Peer Upstream*, o LSR libera o label *upstream* para a FEC e, propaga a mensagem de *Release* para o LDP *Peer DownStream* da FEC.

Se o LSP estiver no estado *Established* e, o LSR receber uma mensagem de *Withdraw* de um LDP *Peer DownStream*, o LSR libera o label *downstream*, envia uma mensagem de *Release* para o *Peer DownStream* que enviou o *Withdraw*, confirmando a liberação do label *downstream* e, propaga a mensagem de *Withdraw* para os LDP *Peers UpStream* à FEC.

As outras transições de estado são semelhantes ao diagrama da subseção anterior. Se o LSP estiver no estado *Established* e, o LSR detectar internamente uma mudança de próximo *hop* para a FEC, ele analisa a condição : *O LSR usa modo de retenção de label Liberal?* e, *O LSR recebeu e manteve um label mapping enviado pelo novo Next-Hop?*

Desta forma, se as duas condições forem verdadeiras, o LSR comporta-se como se tivesse acabado de receber uma mensagem de mapeamento do próximo *hop* novo e realiza a troca de próximo *hop* executando o procedimento de *LOCAL REPAIR*. O LSP continua no estado *Established*.

Se o LSR estiver operando no modo de retenção de label conservativo, ele provavelmente não tem armazenado informações sobre um mapeamento enviado do próximo *hop* novo, já que no modo conservativo, somente os *Mappings* referentes a FECs reconhecidas pelo LSR são armazenados. O LSR, se conservativo,ou liberal, mas que não tenha recebido um mapeamento do próximo *hop*, envia uma mensagem de requisição de label para o próximo *hop* novo. O LSP passa para o estado *New NH Response Awaited*.

Estando no estado *New NH Response Awaited*, ao receber uma mensagem de mapeamento do próximo *hop* novo, o LSR realiza o procedimento de *LOCAL REPAIR* e, o LSP

vai para o estado *Established.* O label *downstream* recebido no mapeamento do próximo *hop* novo é associado ao label *upstream.* O LSR envia também, uma mensagem de *Release* para o *Antigo Next-Hop.*

Estando no estado *New NH Response Awaited,* ao receber uma mensagem de *Release* de um LDP *Peer UpStream,* o LSR libera o label *upstream* para a FEC e, propaga o *Release* para o antigo próximo *hop.* O LSP passa do estado *New NH Response Awaited* para *Idle*

Já ao receber uma mensagem de *Withdraw,* o LSR libera o label *downstream* e propaga a mensagem de *Withdraw* para o LDP *Peer UpStream.* O LSP passa do estado *New NH Response Awaited* para *Idle.*

4.2.3 Diagrama da Máquina de Estado do LDP para LSRs operando no Modo de Distribuição de Label DownStream

Esta subseção apresenta, figura 4.3, a máquina de estado do LDP para LSRs operando no modo de distribuição de label *DownStream.* Esta máquina de estado só apresenta os estados *Idle* e *Established* porque, a máquina de estado da figura 4.3, procura analisar o comportamento do LSR e a mudança do estado do LSP, quando o LSR recebe um mapeamento para uma FEC que não foi requisitada anteriormente.

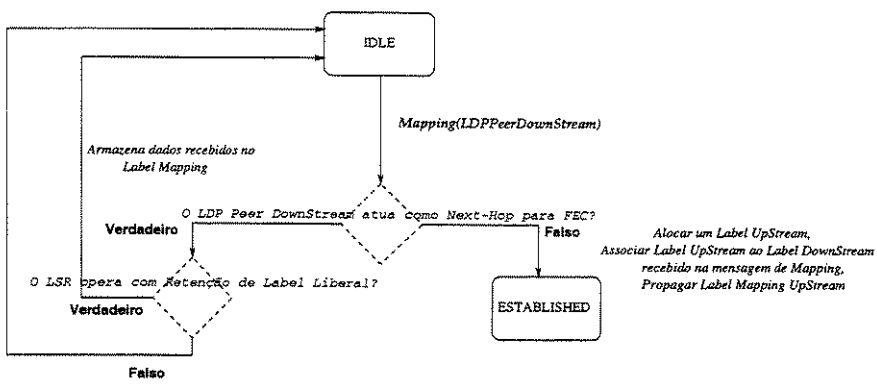

Figura 4.3: Diagrama da Máquina de estado do LDP para LSRs operando no Modo de Distribuição de Label *DownStream*

A ação da máquina de estado para os outros eventos e estados, segue as máquinas definidas nas figuras 4.1 ou 4.2, dependendo do tipo de controle LSP, Independente ou Ordenado.

De acordo com a figura 4.3, se o LSP estiver no estado *Idle* e receber uma mensagem de mapeamento de um *Peer DownStream* para uma FEC que não foi requisitada, o LSR analisa a condição : *O LDP Peer Downstream é Next-Hop para a FEC?*

Se o LDP *Peer DownStream* que enviou o mapeamento for um próximo *hop* para a FEC e, como o LSR está operando no modo de distribuição de label *DownStream*, ele aloca um label *upstream*, associa o label *upstream* alocado ao label *downstream* recém recebido e, propaga a mensagem de mapeamento para os seus LDP *Peers UpStream*. O LSP muda do estado *Idle* para o estado *Established*.

Entretanto, se o LDP *Peer DownStream* que enviou a mensagem de mapeamento, não for um próximo *hop* para nenhuma das FECs reconhecidas pelo LSR, ele analisa outra condição : *O LSR opera com modo de Retenção de Label Liberal?*

Caso o LSR opere com modo de retenção de label liberal, ele armazenará os dados recebidos na mensagem de mapeamento. Caso contrário, ele não armazenará os dados. O LSP continua no estado *Idle*.

4.2.4 Diagrama da Máquina de Estado do LDP para LSRs operando no Modo de Distribuição de Label DownStream-On-Demand

Esta subseção apresenta, figura 4.4, a máquina de estado LDP para LSRs operando no modo de distribuição de label *DownStream-On-Demand*. Esta máquina de estado só apresenta os estados *Idle* e *Established* porque, a figura 4.4 procura analisar o comportamento do LSR, quando ele recebe um mapeamento para uma FEC que não foi requisitada.

A ação da máquina de estado para os outros eventos e estados, segue as máquinas definidas nas figuras 4.1 ou 4.2, dependendo do tipo de controle LSP, Independente ou Ordenado.

Se o LSP estiver no estado *Idle* e ocorrer o evento do recebimento de uma mensagem de mapeamento de um LDP *Peer Downstream*, o LSR analisa a condição : *O LDP Peer DownStream é Next-Hop para a FEC?*

Se o *Peer Downstream* que enviou a mensagem de mapeamento for um próximo *hop* para a FEC, o LSR analisa outra condição : *Existe um Request Pendente para a FEC?*.

Caso exista um pedido de requisição de label pendente, o LSR aloca um label *upstream*, associa o label *upstream* com o label recém recebido no mapeamento, propaga o Mapping *UpStream* e vai para o estado *Established*. Se não existir uma requisição de label para a FEC, o LSR ignora o recebimento do mapeamento e, o LSP mantém-se no estado *Idle*.

Se o LDP *Peer DownStream* não for um próximo *hop* para a FEC associada ao LSP, e não for um próximo *hop* para nenhuma das FECs reconhecidas pelo LSR, o LSR analisa outra condição : *O LSR opera com modo de Retenção de Label Liberal?*

Se o LSR estiver operando no modo de retenção de label Liberal, ele armazena os dados recebidos no mapeamento, caso contrário não armazena os dados e, o LSP continua no estado *Idle*.

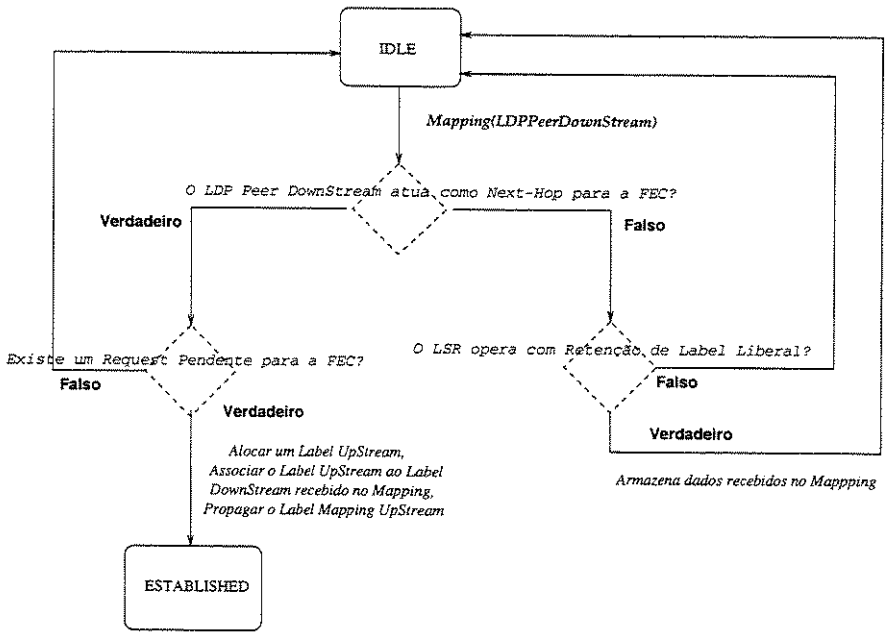

Figura 4.4: Diagrama da Máquina de Estado do LDP para LSRs operando no Modo de Distribuição de Label*DownStream-On-Demand*

4.3 Especificação em SDL dos Procedimentos de Distribuição de Label e da Máquina de Estado do LDP

A especificação apresentada nesta seção, procura especificar formalmente, através da linguagem de especificação formal SDL, um LSR que possa, através da configuração de parâmetros, operar em qualquer um dos modos de distribuição de label, qualquer um dos modos de controle LSP e, qualquer um dos modos de retenção de labels.

Para isto, a especificação segue os procedimentos de distribuição de labels propostos em [21]. De acordo com [21] há quatro procedimentos de distribuição de labels possíveis : *DownStream Unsolicited Independent Control, DownStream Unsolicited Ordered Control, DownStream-On-Demand Independent Control* e *DownStream-On-Demand Ordered Control.* O LSR especificado não possui capacidade de executar *merge.*

Os algoritmos descritos em [21], não definem as estruturas de dados que devem ser usadas para armazenar informações sobre a atividade do protocolo, mas apresentam informações detalhadas, sobre qual informação deve ser armazenada e recuperada.

A especificação apresentada nesta seção, utilizou a estrutura de dados de *LSP Control Block* descrita em [24], para armazenar as informações sobre os LSPs a serem estabelecidos. Para cada LSP a ser estabelecido é criada uma entrada na tabela de blocos de controle LSP. As máquinas de estado apresentadas na seção 4.2 estão embutidas na especificação, porque o estado da máquina do LDP é armazenado em um dos campos da estrutura de

controle de LSPs, subseção 4.3.1.

O objetivo é especificar formalmente as informações textuais contidas em [24] e [21]. Os procedimentos de distribuição de label do *internet-draft* de especificação do LDP [21], não especifica uma máquina de estado do LDP que foi então proposta em [24]. A especificação procura então, combinar os procedimentos da especificação do LDP, com a estrutura de controle de LSP proposta pela máquina de estado de [24]. Procura-se analisar neste trabalho, além do comportamento funcional do LSR diante do recebimento de mensagens LDP, a mudança de estados da máquina do LDP para cada LSP a ser estabelecido.

As estruturas de dados, os procedimentos, os parâmetros de configuração, e os sinais trocados entre o ambiente e o sistema, são descritos na subseção 4.3.1.

4.3.1 Especificação das Variáveis, Estruturas e Sinais do Sistema MPLS

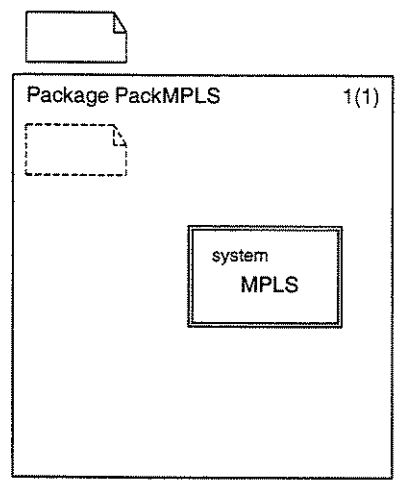

Figura 4.5: Pacote PackMPLS (Sistema MPLS)

Esta subseção apresenta a descrição do sistema especificado, as variáveis e estruturas criadas, a definição dos sinais e listas de sinais, assim como os diagramas SDL do sistema especificado.

A figura 4.5 mostra a estrutura de package criada contendo o sistema MPLS.

A figura 4.6 apresenta uma visão do *Organizer(SDT)* do sistema especificado ao qual foi atribuído o nome MPLS, representando os procedimentos de distribuição de label do LDP. Foi criada uma estrutura package, *PackMPLS*, que contém o sistema *MPLS*, figura

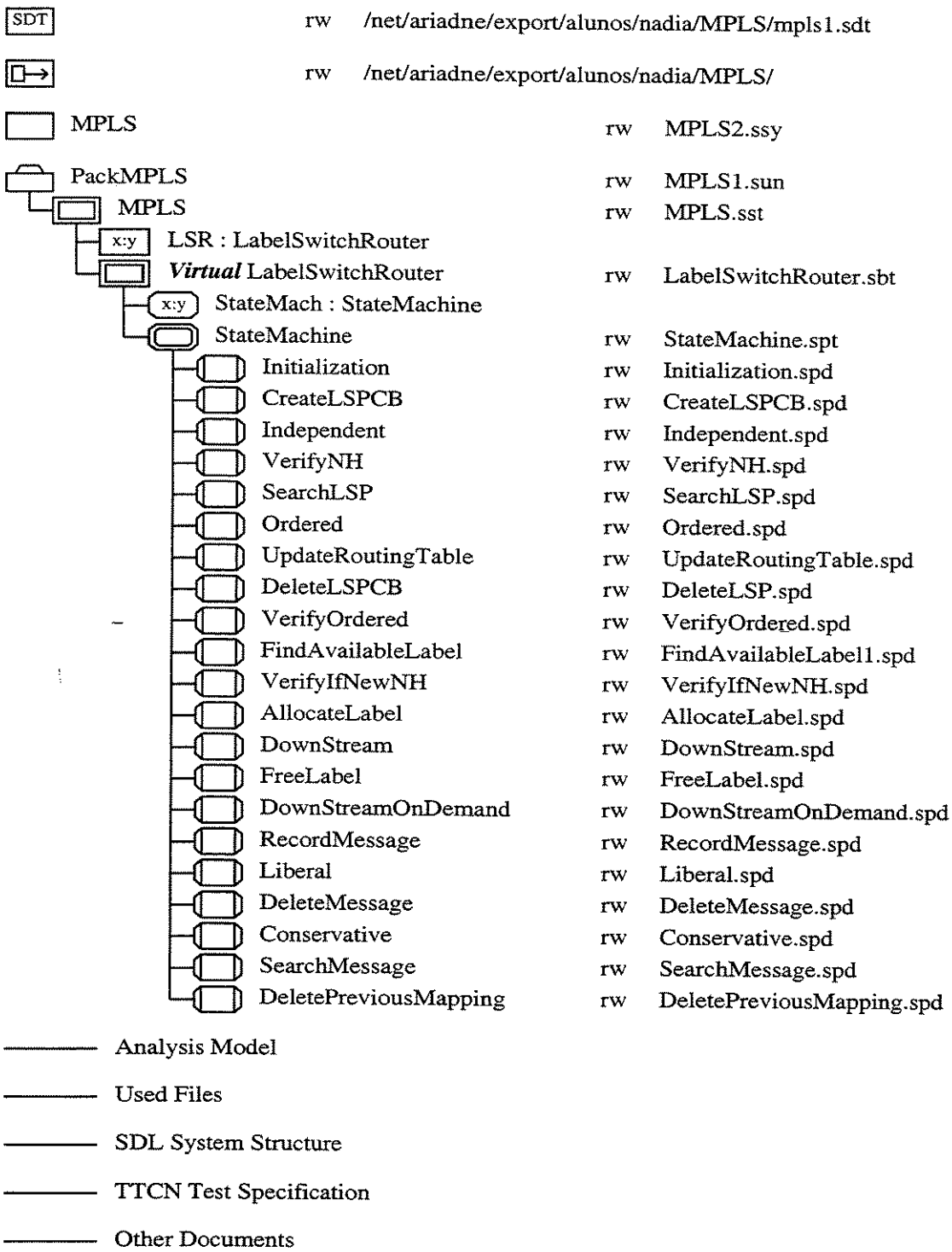

Figura 4.6: Visão do Organizer(SDT) do Sistema MPLS

4.5, para que em eventuais trabalhos futuros toda a especificação do sistema possa ser herdada. Seguindo-se a estrutura hierárquica do SDL, pode-se observar na figura 4.6, que o sistema MPLS possui um processo de nome *LabelSwitchRouter*, que contém por sua vez, o processo *StateMachine*. O processo *StateMachine* possui 22 procedimentos, onde o primeiro procedimento é o de nome *Initialization* e, o último procedimento é o de nome *VerifyMapping*.

A figura 4.7, apresenta a definição dos sinais do sistema MPLS. O envio de cada um dos sinais definidos neste diagrama, representa a ocorrência de um evento como por exemplo, o sinal *LDPRequest* representa o envio de uma mensagem de requisição de label de um LDP *Peer UpStream* para o LSR especificado.

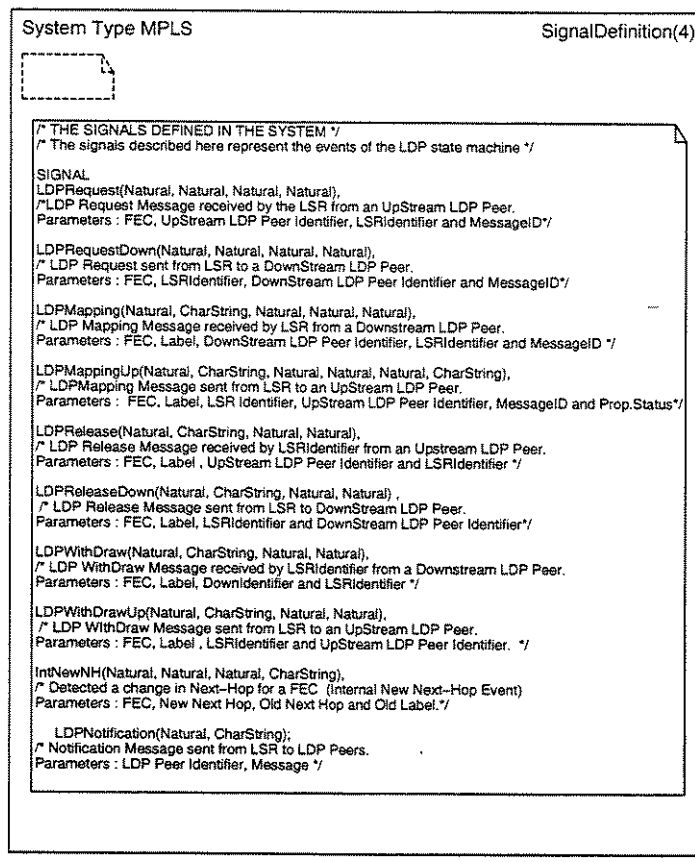

Figura 4.7: Definição dos Sinais do Sistema MPLS

Os possíveis eventos, são então representados pelo envio dos sinais *LDPRequest*, *LDP-Mapping*, *LDPRelease*, *LDPWithDraw*, *LDPNotification* e, *IntNewNH*. Foram definidos também sinais de mesmo nome seguidos do sufixo *Down* representando o envio do sinal

para um *Peer DownStream* ou, *Up* representando o envio do sinal para um *Peer UpStream*. Estes sinais representam os mesmos eventos dos sinais de mesmo nome, porém sem o sufixo. Entretanto, estes sinais com sufixos, foram criados para representar e diferenciar os sinais enviados pelo LSR especificado ao ambiente externo, dos sinais enviados do ambiente externo (representando LDP *peers UpStream* e *DownStream*) para o LSR.

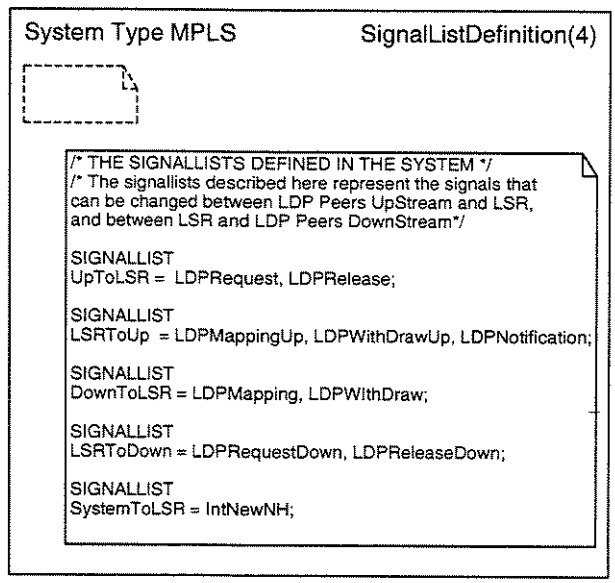

Figura 4.8: Definição das Listas de Sinais do Sistema MPLS

A figura 4.8 apresenta a definição de listas de sinais que são usadas para definir quais os sinais que podem trafegar em um canal, além de simplificarem o entendimento dos diagramas SDL. Os possíveis sinais que podem ser enviados do ambiente externo para o LSR e do LSR para o ambiente externo, são divididos em listas de sinais declaradas no diagrama da figura 4.8. A lista de sinais *UpToLSR*, representa os sinais que podem ser enviados de um LDP *Peer UpStream* para o LSR especificado. A lista de sinais *LSRToUp*, representa os sinais que podem ser enviados do LSR especificado para os LDP *Peers UpStream*. A lista *DownToLSR*, representa os sinais que podem ser enviados dos LDP *Peers DownStream* para o LSR especificado. A lista de sinais *LSRToDown*, representa os sinais que podem ser enviados do LSR especificado para os LDP *Peers DownStream*. E finalmente, a lista *SystemToLSR* representa os sinais enviados pelo embiente externo ao

LSR, e que sinalizam uma mudança de configuração do sistema. É importante ressaltar que os LDP *Peers UpStream* e *DownStream* são representados pelo envio de sinais do ambiente externo em diferentes canais.

A definição das listas de sinais descritas na figura 4.8, foram criadas para especificar quais sinais podem ser enviados do ambiente ao sistema, e ainda, por quais canais. Esta representação procura caracterizar o sentido de distribuição de labels pelo LDP. Os labels são associados a FECs pelos LDP *Peers DownStream* e propagados no sentido *DownStream-To-UpStream* e, as requisições de labels do sentido *UpStream-To-DownStream.*

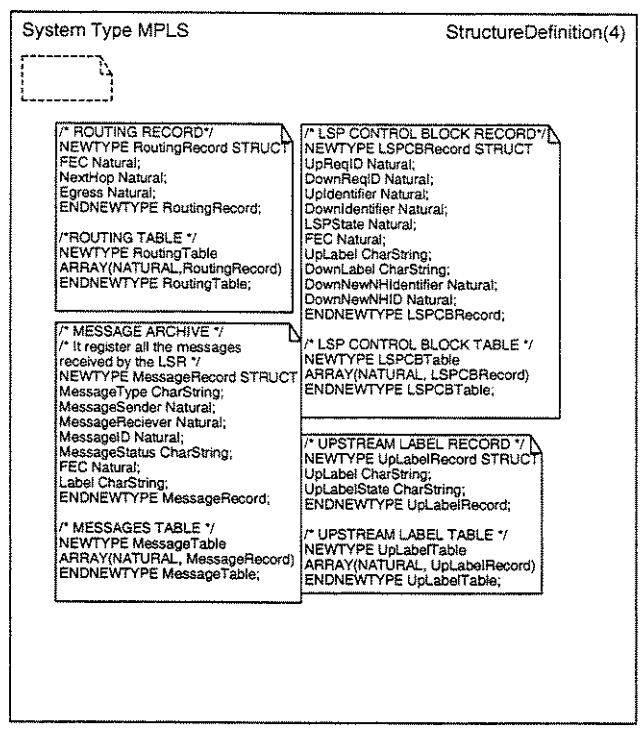

Figura 4.9: Definição das Estruturas de Dados (Sistema MPLS)

A figura 4.9 apresenta a definição das estruturas de dados definidas no sistema MPLS. As estruturas declaradas estão divididas em quatro quadros. Cada quadro apresenta duas declarações *STRUCT*. O primeiro *STRUCT* é a declaração de uma estrutura de dados (*Record*) e, o segundo *STRUCT* é a declaração de um vetor (*Table*), onde cada elemento

do vetor é uma estrutura de dados do primeiro *STRUCT*.

O primeiro quadro, define uma estrutura que simboliza uma tabela de roteamento do LSR e que armazena a FEC, o próximo *hop* para a FEC e, se o LSR é nó de saída da rede ou não em relação a esta FEC.

O segundo quadro, define uma estrutura para a tabela de mensagens, que tem a função de registrar as mensagens LDP recebidas e/ou enviadas pelo LSR aos seus LDP *peers*. Sua estrutura, *MessageRecord*, apresenta 7 campos que armazenam respectivamente o tipo da mensagem, o identificador do *peer* que enviou a mensagem, o identificador do *peer* que recebeu a mensagem, o identificador da mensagem, o status da mensagem e, o Label transmitido na mensagem, se existir.

O terceiro quadro, apresenta a definição da estrutura de controle de LSPs. Para cada LSP a ser estabelecido, é criado um registro da estrutura *LSPCBRecord*, como em [24] onde é definido um *LSP Control Block* para cada LSP a ser estabelecido. Nesta especificação, a criação de um *LSP Control Block* para cada FEC na qual se procura estabelecer um LSP, é representada pela criação de um registro na tabela de LSPCBs. Cada registro de LSPCB representa uma máquina de estado, onde o estado da máquina, apresentado nos diagramas da seção 4.2, é armazenado em um campo numérico da estrutura LSPCB.

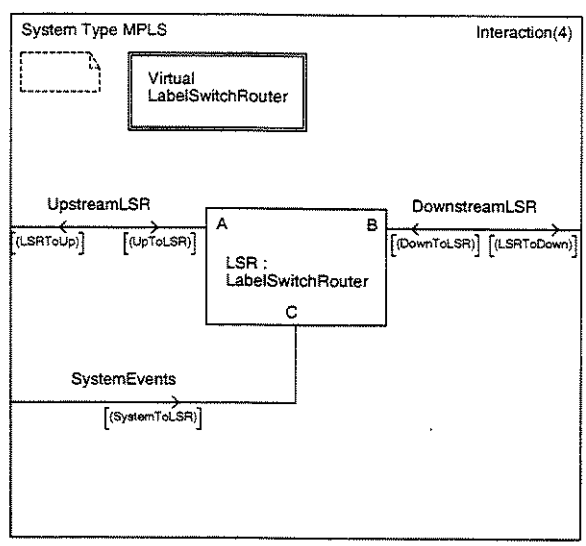

Figura 4.10: Sistema MPLS

A estrutura de controle LSP definida neste capítulo foi baseada na estrutura proposta por [24]. De acordo com [24], um bloco de controle LSP possui 10 campos que armazenam respectivamente : o identificador da mensagem de requisição de label enviada por um *peer*

UpStream, o identificador de mensagem de requisição de label enviada pelo LSR para um *peer DownStream*, o identificador do *peer UpStream*, o identificador do *peer DownStream*, o **ESTADO** do LSP, a FEC associada ao LSP, o label *UpStream* (alocado pelo LSR), o label *DownStream* (alocado pelo LDP *Peer DownStream*), o identificador do próximo *hop* novo e, o identificador da mensagem de requisição de label para o próximo *hop* novo. O campo de definição do comportamento do LSR em relação ao caso de detecção de um próximo *hop* novo proposto em [24], foi eliminado pois o LSR da especificação está configurado para sempre executar o procedimento de *Local Repair*.

Finalmente o quarto quadro, define uma estrutura para uma tabela de labels *UpStream* do LSR. A tabela de labels *upstream* representa os labels *upstream* do LSR disponíveis para o uso. A tabela armazena no primeiro campo , *UpLabel*, o nome do label e, no segundo campo, *UpLabelState*, o estado do label. Aqui, um label é representado por uma string.

A figura 4.10 apresenta a especificação do sistema MPLS. O sistema MPLS, figura 4.6, é formado pelo bloco *LabelSwitchRouter* que representa um LSR. Somente uma instância deste bloco foi criada, *LSR* porque, o objetivo da espercificação é analisar o comportamento de um LSR em relação ao recebimento de mensagens LDP e à construção de blocos de controle de LSPs.

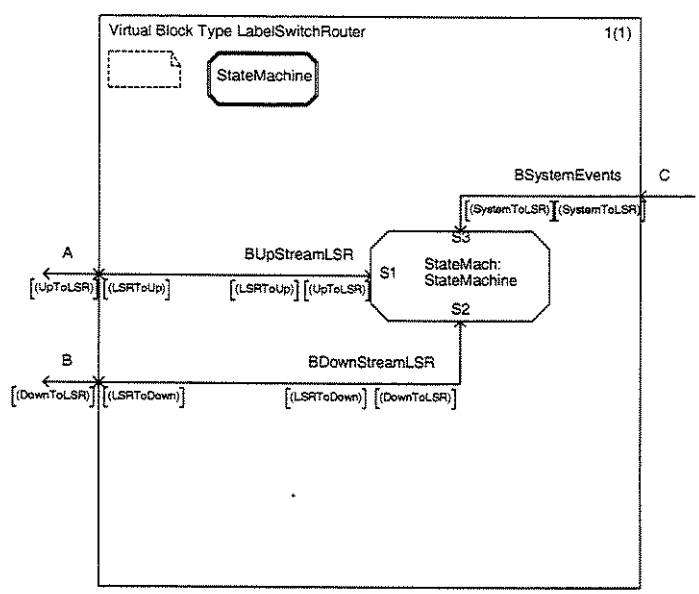

Figura 4.11: Bloco LabelSwitch Router (Sistema MPLS)

A instância *LSR*, figura 4.10, apresenta três canais de nomes : *UpstreamLSR*, *DownstreamLSR* e *SystemEvents*, que ligam a instância LSR ao ambiente externo.

O canal *UpstreamLSR* representa a comunicação do LSR especificado com todos os seus LDP *Peers UpStream*. As listas de sinais *UpToLSR* e *LSRToUp*, definidas na figura 4.8, caracterizam os sinais que podem ser enviados de um LDP *Peer UpStream* para o LSR ou, do LSR para um LDP *Peer UpStream*. Da mesma forma, o canal *DownstreamLSR* representa a comunicação do LSR com seus LDP *Peers DownStream* e, as listas de sinais *LSRToDown* e *DownToLSR*, caracterizam os sinais que podem ser trocados entre o LSR e seus LDP *Peers DownStream*.

O canal *SystemEvents* foi criado para representar a ocorrência do evento de detecção interna de mudança de próximo *hop* para uma FEC. Este evento, assim como os eventos de recebimento de mensagens LDP, é representado pelo envio de um sinal, neste caso o sinal *IntNewNH* declarado na lista de sinal *SystemToLSR*, figura 4.8.

A figura 4.11 apresenta a especificação do bloco *Label Switch Router* que possui um processo de nome *StateMachine*. O bloco *LabelSwithRouter* foi declarado como *Virtual Block Type* para que possa ser reutilizado em outros trabalhos.

O processo *StateMachine* é o responsável por descrever o comportamento do LSR diante do recebimento de mensagens do LDP. Na figura 4.11, pode-se observar a criação de uma instância do processo *StateMachine*, de nome *StateMach*, que se comunica com o ambiente externo através dos gateways A, B e C. As rotas *BUpStreamLSR*, *BDownStreamLSR* e *BSystemEvent* representam os canais *UpstreamLSR*, *DownstreamLSR* e *SystemEvents* do sistema MPLS apresentados na figura 4.10, porém são de visão interna ao bloco *LabelSwitchRouter*.

4.3.2 Especificação do Processo StateMachine

Esta subseção apresenta a especificação formal do processo *StateMachine*, que é o responsável por definir o comportamento funcional do LSR diante do recebimento dos sinais definidos no sistema, assim como a geração das máquinas de estado do LDP, definida nos diagramas da seção 4.2, para cada LSP a ser estabelecido.

A especificação do processo *StateMachine* está dividida em 8 diagramas SDL. O primeiro diagrama, figura 4.12, apresenta a declaração das variáveis utilizadas no processo. O segundo diagrama, figura 4.13, apresenta os procedimentos presentes no processo *StateMachine*.

Os outros seis diagramas, figuras 4.14, 4.15, 4.16, 4.17, 4.18 e 4.19 descrevem o comportamento do processo *StateMachine* diante do acontecimento dos possíveis eventos que são representados na especificação, através do envio de sinais do ambiente externo ao sistema :

Figura 4.14 - recebimento do sinal *LDPRequest*,

Figura 4.15 - recebimento do sinal *LDPMapping*,

Figura 4.16 - recebimento do sinal *LDPRelease*,

Figura 4.17 - continuação das ações do LSR em relação ao recebimento do sinal *LDPRelease*,

Figura 4.18 - recebimento do sinal *LDPWithDraw* e,

Figura 4.19 - recebimento do sinal *IntNewNH*.

4.3.2.1 Variáveis e Procedimentos do Processo StateMachine

Na figura 4.12 é apresentado o diagrama SDL de declaração das variáveis e estruturas utilizadas no processo *StateMachine*. O primeiro quadro mais a esquerda, apresenta a declaração das variáveis que permitem que o LSR seja configurado (*Configuration Parameters*). A variável *LabDisProc* configura o tipo de procedimento de distriuição de labels. Os procedimentos especificados neste trabalho são baseados nos procedimentos definidos em [15]. Os valores permitidos para a variável *LabDisProc* são : 1 - *DownStream* de Controle Independente, 2 - *DownStream* de Controle Ordenado, 3 - *DownStream-On-Demand* de Controle Independente e, 4 - *DownStream-On-Demand* de Controle Ordenado, referenciados em [15] como os procedimentos de nome *PushUnconditional*, *PushConditional*, *PulledUnconditional* e *PulledConditional* respectivamente.

A variável *LSRIdentifier*, armazena o endereço IP identificador do LSR especificado. A variável *LabelRetention* configura o tipo de rentenção de label a ser usado no LSR. Configurando-se a variável *LabelRetention* com o valor 1, O LSR opera com retenção de label Conservativo e, com o valor 2, o LSR opera com retenção de label Liberal.

A variável *UpLDPPeer* armazena o endereço IP identificador do LDP *Peer UpStream*. Esta especificação está considerando que há apenas um LDP *Peer UpStream* ao LSR, para facilitar a operação do LSR especificado e o, envio de mensagens para os LDP *Peers UpStream*.

O segundo quadro à esquerda na figura 4.12 (*Parameters Recieved in Signals*), apresenta a declaração de variáveis auxiliares que, têm a função de receber os parâmetros dos sinais enviados.

No terceiro quadro da figura 4.12, à direita na parte superior (*From LSPCB Record*), há a descrição dos valores dos estados possíveis de um LSP, isto é, os estados possíveis da máquina de estados do LDP definida na seção 4.2. O estado é representado por um número inteiro que varia de 1 a 4 : 1 - *Idle*, 2 - *Response Awaited*, 3 -*Established* e, 4 - *New NH Response Awaited*.

Logo abaixo dos possíveis estados de um LSP na figura 4.12, há a descrição dos valores que o status de um label UpStream pode assumir : A - *Available* e U - *In Use*.

A seguir, na figura 4.12, apresenta-se a declaração das variáveis onde os tipos são as tabelas definidas na figura 4.9 : *Routing* do tipo *RoutingTable*, *LSPCB* do tipo *LSPCBTable*, *UpLabel* do tipo *UpLabelTable* e, *Message* do tipo *MessageTable*.

As variáveis *RoutingNumber*, *LSPCBNumber* e *MessageNumber*, armazenam o número de elementos nas tabelas *Routing*, *LSPCB* e *Message* respectivamente. A variável *MaxUpLabel* configura o número máximo de labels *Upstream* que podem ser alocados pelo LSR.

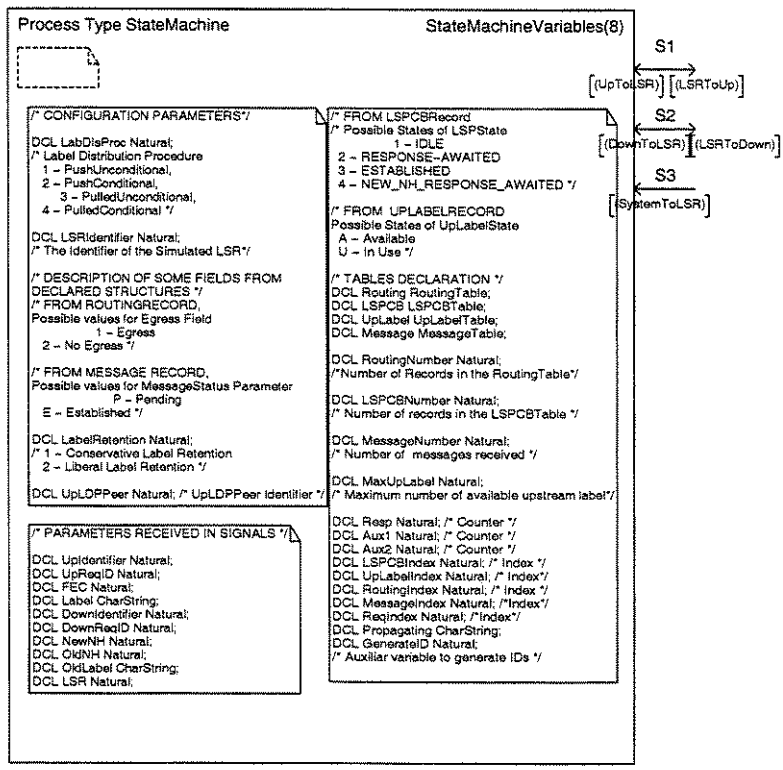

Figura 4.12: Definição das Variáveis do Processo StateMachine (Bloco LabelSwithRouter)

Ainda na figura 4.12 as variáveis *Resp, Aux1* e *Aux2* são variáveis auxiliares. As variáveis de nome terminando em *Index* armazenam índices de tabelas. A variável *GenerateID*, é uma variável numérica que gera valores que são atribuídos a identificadores de mensagens enviadas pelo LSR aos seus *peers*.

A figura 4.13 apresenta a lista dos procedimentos pertencentes ao processo *StateMachine* e, o código em SDL que o inicializa. Pode-se observar no canto direito inferior da figura 4.13, um símbolo de *start* onde a próxima transição é a chamada ao procedimento *Initialization*. Após a chamada ao procedimento *Initialization*, o processo *StateMachine* está pronto para receber sinais permanecendo no estado *Waiting*.

A seguir apresenta-se uma descrição resumida da função de cada um dos procedimentos do processo *StateMachine*, na ordem em que aparecem na figura 4.13.

Procedimento *Initialization* : O procedimento *Initialization* é responsável por inicializar as variáveis de configuração do LSR, figura 4.12. Além disso, este procedimento inicializa a tabela de roteamento do LSR, *Routing*, com informações das FECs as quais o LSR possui um próximo *hop*. Considera-se nesta especificação uma FEC sendo um endereço de prefixo IP. A tabela de labels *Upstream* do LSR, *UpLabel*, é inicializada configurando-se o número máximo de labels *upstream*. No estado inicial do programa, todos os labels são considerados disponíveis para serem alocados.

Procedimento *VerifyNH*: O procedimento *VerifyNH* tem a função de, dado uma FEC, realizar uma busca na tabela de roteamento do LSR (*Routing*), para verificar se há um próximo *hop* para a FEC fornecida. O procedimento recebe como parâmetro, a FEC e, retorna um valor numérico. O valor retornado é igual a 0 (zero) caso não haja um próximo *hop* para a FEC ou, é igual ao índice da entrada na tabela de roteamento do registro que possua um próximo *hop* para a FEC.

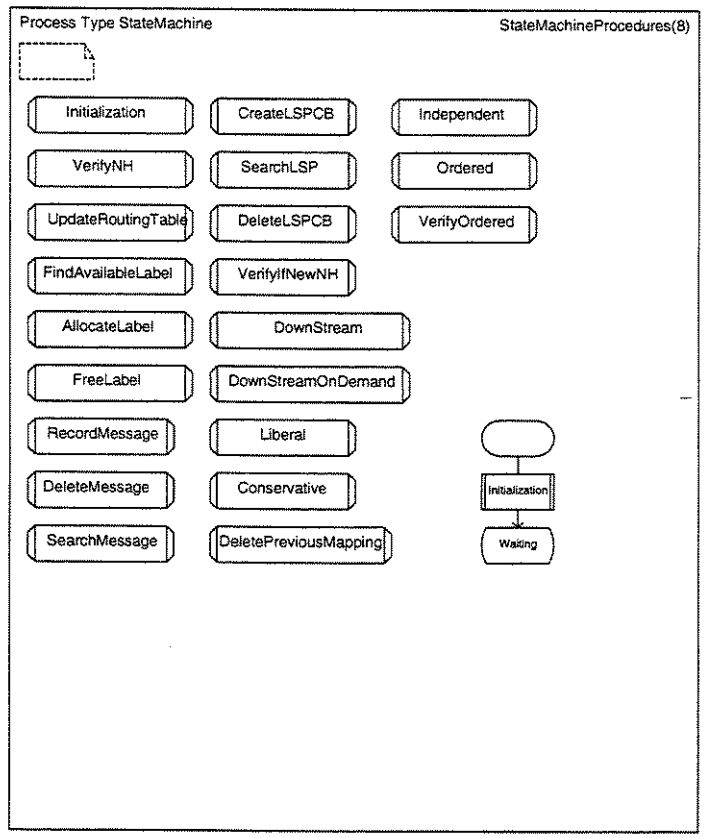

Figura 4.13: Definição dos Procedimentos do Processo StateMachine (Bloco LabelSwithRouter)

Procedimento *Update Routing Table* : O procedimento *Update Routing Table* tem a função de atualizar dados da tabela de roteamento do LSR (*Routing*). Este procedimento é ativado quando o sistema recebe o sinal *IntNewNH* do ambiente. O envio deste sinal representa a detecção pelo LSR de uma mudança de próximo *hop* para uma determinada FEC. O procedimento atualiza

o registro da tabela de roteamento dados os valores da FEC e do próximo *hop* novo recebidos no sinal *IntNewNH*.

Procedimento *Find Available Label* : O procedimento *Find Available Label* tem a função de realizar uma busca na tabela de labels *upstream* do LSR, *UpLabel*, para verificar se há um label disponível para ser alocado. Este procedimento é executado toda vez que o LSR precisa alocar um label para enviar em uma mensagem *LDPMapping* a um LDP *Peer UpStream*.

Procedimento *Allocate Label* : O procedimento *Allocate-Label* tem a função de alterar o status de um label de *Available* para *Used*. Esta operação indica que o label não está mais disponível, mas está alocado. Este procedimento é executado toda vez que o LSR aloca um label que será propagado em uma mensagem *LDPMapping* para um LDP Peer UpStream.

Procedimento *Free Label* : O procedimento *Free Label* é responsável por alterar o status de um label de *Used* para *Available*. Esta operação indica que o label que estava alocado, agora está disponível. Este procedimento é executado toda vez que o LSR recebe um sinal *LDPRelease* de um LDP *Peer UpStream*.

Procedimento *Record Message* : O procedimento *Record Message* é responsável por gravar os dados de uma mensagem LDP recebida e/ou enviada pelo LSR a um LDP *Peer UpStream* ou a um LDP *Peer DownStream*, na tabela de mensagens *Message*. Este procedimento é executado quando o LSR recebe ou envia uma mensagem LDP. Os parâmetros passados ao procedimento *Record-Message* são : tipo da mensagem, identificador do *peer* que originou a mensagem, identificador do *peer* que recebeu a mensagem, o identificador da mensagem, a FEC associada à mensagem, o label associado à mensagem e o status da mensagem.

Procedimento *Delete Message* : O procedimento *Delete Message* tem a função de remover uma mensagem LDP registrada na tabela de mensagens, *Message*. Este procedimento é executado, por exemplo, para remover da tabela de mensagens os dados referentes a uma requisição de label que já foi atendido, isto é, uma requisição de label que possuía o status pendente, mas que recebeu um mapeamento de label. O procedimento *Delete Message* recebe como parâmetro, o índice do registro na tabela de mensagens que precisa ser removida.

Procedimento *Search Message* : O procedimento *Search Message* tem a função de procurar pelos dados de uma mensagem na tabela de mensagens, *Message*. A mensagem é procurada passando-se como parâmetros o tipo da mensagem, o identificador de quem originou a mensagem, o identificador de quem recebeu a mensagem e, a FEC associada à mensagem LDP. O procedimento retorna 0 (zero) caso a mensagem não seja encontrada ou, o índice do registro da tabela de mensagens que atenda aos parâmetros passados ao procedimento.

Procedimento *Create LSPCB* : O procedimento *Create LSPCB* tem a função de inserir um novo registro de bloco de controle LSP na tabela de LSPs, *LSPCB*. Os parâmetros passados ao procedimento são : Identificador da mensagem de requisição de label enviada pelo LDP *Peer UpStream*, identificador da mensagem de requisição de label enviada pelo LSR ao LDP *Peer DownStream*, o identificador do LDP *Peer UpStream*, o identificador do LDP *Peer DownStream*, o estado do LSP, a FEC associada ao LSP, o label *upstream*, o label *downstream*, identificador do próximo *hop* novo para a FEC e, o identificador da mensagem de requisição de label para o próximo *hop* novo. Este procedimento é executado toda vez que se recebe uma requisição de label para uma FEC que ainda nao está associada a um LSP. E ainda, toda vez que o LSR estiver operando no modo de distribuição *DownStream* e, receber um mapeamento de label de um LDP *Peer DownStream*.

Procedimento *Search LSP* : O procedimento *Search LSP* tem a função de procurar por um bloco de controle LSP na tabela de LSPs. A busca é realizada de acordo com os parâmetros que são passados ao procedimento. Há três tipos de buscas possíveis, FEC e identificador do *request downstream*, FEC e o identificador do LDP *Peer UpStream* e, FEC e o identificador do LDP *Peer DownStream*.

Procedimento *Delete LSPCB* : O procedimento *Delete LSPCB* tem a função de remover um registro de LSP da tabela *LSPCB*, dado o seu índice. Este procedimento é executado quando o LSR recebe um *LDPRelease* para uma FEC de um LDP *Peer UpStream*.

Procedimento *Verify If New NH* : O procedimento *Verify If New NH* tem a função de verificar se um mapeamento recebido está sendo enviado de um próximo *hop* novo para uma FEC, onde a mudança de próximo *hop* foi detectada pelo LSR. Caso o mapeamento venha de um próximo *hop* novo, o procedimento *Verify If New NH* realiza a operação de *Local Repair*. Os dados do LSP referente à FEC que sofreu mudança de próximo *hop* são atualizados da seguinte forma : o identificador do LDP *Peer DownStream* recebe o identificador do próximo *hop* novo, o identificador da mensagem de *request downstream* recebe o valor do identificador da mensagem de requisição de label previamente enviada pelo LSR ao próximo *hop* novo e, os campos que armazenavam as informações sobre o próximo *hop* novo são reinicializadas com o valor 0 (zero).

Procedimento *DownStream* : O procedimento *DownStream* é executado toda vez que o LSR receber um *LDPMapping* de um LDP *Peer DownStream* e, estiver operando com os procedimentos de distribuição configurados para *DownStream Unsolicited Independent*, ou *DownStream Unsolicited Ordered*. No modo de distribuição *DownStream Unsolicited*, quando o LSR receber um mapeamento de um LDP *Peer DownStream*, ele pode propagar o mapeamento

para seus LDP *Peers UpStream*, mesmo que estes não tenham requisitado um label para a FEC associada. Se já existir um LSPCB para a FEC recebida na mensagem de mapeamento, o procedimento *DownStream* atualiza os valores do campo de label *downstream* com o label recebido na mensagem e, analisa se o mapeamento precisa ou não ser propagado para os LDP *Peers UpStream*. Se o LSR estiver configurado para trabalhar com controle LSP Independente e, se o LSR recebeu previamente um pedido de requisição de label para a FEC que está recebendo um mapeamento, não é preciso propagar o *LDP-Mappping UpStream* já que, o LSR já alocou um label *upstream* e propagou aos seu LDP *Peers UpStream*, no momento do recebimento da requisição de label. Entretanto, se o LSR estiver operando no modo de controle LSP Ordenado e, tiver recebido um *LDPRequest* de um LDP *Peer UpStream*, o pedido ainda não foi atendido. O LSR deve então alocar um label *upstream*, atualizar o estado do LSP de *Response Awaited* para *Established*, atualizar o campo de label *upstream* do LSPCB com o label alocado e, propagar o mapeamento para o LDP *Peer* requisitante. Caso não exista um LSPCB para a FEC recebida na mensagem de mapeamento, o LSR cria um novo registro de LSP chamando o procedimento *Create LSPCB*, aloca um label *upstream* e, propaga o mapeamento para o LDP *Peer UpStream*.

Procedimento *DownStream On Demand* : O procedimento *DownStream On Demand* é executado toda vez que um LSR receber uma mensagem LDP Mapping de um LDP Peer DownStream e, estiver configurado para operar com os procedimentos de distribuição de labels DowStream-On-Demand Independente ou *DownStream-On-Demand* Ordenado. No modo de distribuição de labels *DownStream-On-Demand*, o LSR só vai propagar um mapeamento recebido de um LDP *Peer UpStream* se ele já tiver recebido uma requisição de label para a FEC do LDP *Peer UpStream*. Se o LSR já tiver recebido uma requisição de label do LDP *Peer UpStream* para a FEC e estiver operando no modo de controle LSP Ordenado, o LSR atualiza o campo de label *downstream* do LSPCB para a FEC com o label recebido na mensagem de mapeamento, atualiza o estado do LSP de *Response Awaited* para *Established*, aloca um label *upstream* , atualiza o campo de label *upstream* do LSPCB com o label alocado e, propaga o mapeamento para o LDP *Peer UpStream*. Se o LSR estiver operando no modo de controle LSP Independente, o LSR deve ter alocado um label *upstream* e propagado ao LDP *Peer UpStream* no momento em que foi feito a requisição de label. Desta forma, não é necessário propagar o *mapping upstream*. Só é necessário atualizar o campo de label *downstream* do LSPCB com o label recebido na mensagem de mapeamento.

Procedimento *Liberal* : O procedimento *Liberal* é responsável por executar o modo de retenção de label liberal, isto é, os mapeamentos de labels recebidos são registrados pelo LSR, mesmo que o LDP *Peer DownStream* que enviou o mapeamento, não seja um próximo *hop* para as FECs reconhecidas pelo LSR.

O procedimento *Liberal* faz uma chamada ao procedimento *Record Message* e, armazena as informações recebidas na mensagem de mapeamento como, o identificador do LDP que enviou a mensagem, o identificador da mensagem, a FEC e, o label associado à FEC.

Procedimento *Conservative* : O procedimento *Conservative* é responsável por executar o modo de retenção de label conservativo, isto é, o LSR não armazena informações recebidas de *mappings*, cujo LDP que originou a mensagem, não seja um próximo *hop* para uma das FECs reconhecidas pelo LSR. O procedimento *Conservative* envia uma mensagem de *LDPReleaseDown* para o label e FEC transmitidos na mensagem de mapeamento, isto é, uma mensagem de *release* que parte do LSR para o LDP *DownStream* que gerou a mensagem de mapeamento.

Procedimento *Delete Previous Mapping* : O procedimento *Delete Previous Mapping* tem a função de localizar na tabela de mensagens, as mensagens de mapeamento enviadas de um LPD *Peer DownStream* para o LSR, ou do LSR para o LDP *Peer UpStream*, se as mensagens existirem. Caso as mensagens existam, elas são removidas da tabela de mensagens. Este procedimento é executado quando o LSR recebe uma mensagem de *release* de um *peer upstream* para uma determinada FEC.

Procedimento *Independent*: O procedimento *Independent* tem a função de realizar o controle LSP Independente. Ele é executado toda vez que, o LSR operando nos modos de distribuição *DownStream* Independente ou *DownStream-On-Demand* Independente, receber uma mensagem de requisição de label para uma FEC, de um LDP *Peer UpStream*. A operação básica do procedimento *Independent* é alocar um label *upstream*, atualizar o campo de label *upstream* do LSPCB da FEC requisitada, com o label alocado e, enviar um *mapping upstream* com o label alocado, ao LDP *Peer UpStream* requisitante. Além disso, se o LSR não for nó de saída da rede para FEC requisitada ou, se o LSR ainda não recebeu e reteve as informações de um mapeamento do próximo *hop* para a FEC, o procedimento *Independent*, propaga o *request downstream*, enviando uma mensagem *LDPRequestDown* para o próximo *hop* para a FEC.

Procedimento *Ordered* : O procedimento *Ordered* tem a função de realizar o controle LSP Ordenado. Ele é executado toda vez que, o LSR operando nos modos de distribuição *DownStream* Ordenado ou *DownStream-On-Demand* Ordenado, receber uma mensagem de requisição de label para uma FEC de um LDP *Peer UpStream*. O procedimento *Ordered* verifica, através de consultas as tabelas de roteamento e de mensagens, se o LSR é nó de saída da rede para a FEC ou, se o LSR já recebeu e armazenou as informações de um mapeamento do próximo *hop* para a FEC. Caso uma das condições for atendida, o LSR, aloca um label *upstream*, atualiza o campo de label *upstream* do LSP para a

FEC com o label alocado, atualiza o estado do LSP para *Established* e, envia um mapeamento para o requisitante *upstream*. Caso nenhuma das condições forem atendidas, o LSR não aloca e envia o *mapping upstream*, mas marca a requisição de label como *Pendente* e, propaga a requisição de label para o próximo *hop* para a FEC.

Procedimento *Verify Ordered* : O procedimento *Verify Ordered* verifica se existe um pedido de requisição de label pendente para uma FEC, quando o LSR está operando no modo de controle LSP Ordenado.

4.3.2.2 Especificação em SDL - Processo *StateMachine* (Recebimento de Mensagem de *Request*)

A figura 4.14 representa o comportamento do LSR diante da ocorrência do evento de recebimento de uma mensagem de requisição de label do LDP. Na especificação do processo *StateMachine*, este evento é representado pelo envio do sinal *LDPRequest* do ambiente externo ao sistema MPLS. O sinal *LDPRequest* é enviado através do canal *UpStreamLSR*, no nível entre blocos, figura 4.10 e, através da rota *BUpStreamLSR*, no nível entre processos, figura 4.11. O sinal *LDPRequest* está definido na lista de sinais *UpToLSR*, figura 4.8.

Seguindo-se as estruturas do diagrama SDL da figura 4.14, estando o processo *StateMachine* no estado *Waiting*, ao receber um sinal *LDPRequest*, o procedimento *VerifyNH* é chamado para verificar se o LSR possui um próximo *hop* para a FEC requisitada. Caso não exista um próximo *hop* para a FEC, o processo *StateMachine* envia o sinal *LDPNotification* com a mensagem *No Route* para o LDP *Peer UpStream* (*UpIdentifier*), que enviou a mensagem de requisição de label. Caso exista um próximo *hop* para a FEC, o processo faz uma chamada ao procedimento *SearchMessage* para verificar se já foi recebida uma mensagem de requisição de label anteriormente para a mesma FEC.

Se já existir uma mensagem de requisição de label registrada na tabela de mensagens e, se o identificador da mensagens registrada for igual ao identificador da mensagem de requisição de label recebida, o LDP *Peer UpStream* enviou um pedido duplicado e, o processo *StateMachine* ignora a mensagem de requisição de label. Como o LSR não é capaz de realizar *merge*, ele deve saber distinguir pedidos repetidos, de pedidos de vários *peers upstream* para a mesma FEC. Caso o LSR não tenha recebido previamente uma mensagem de requisição de label para a FEC, o processo faz uma chamada ao procedimento *CreateLSPCB* para criar um bloco de controle para o estabelecimento de um novo LSP para a FEC.

Se o LSR estiver operando no modo de controle LSP Independente (*LabDisProc = 1 ou 3*), o processo *StateMachine* faz uma chamada ao procedimento *Independent*, caso contrário, faz uma chamada ao procedimento *Ordered*.

O processo *StateMachine* retorna ao estado *Waiting* pronto para receber o próximo sinal.

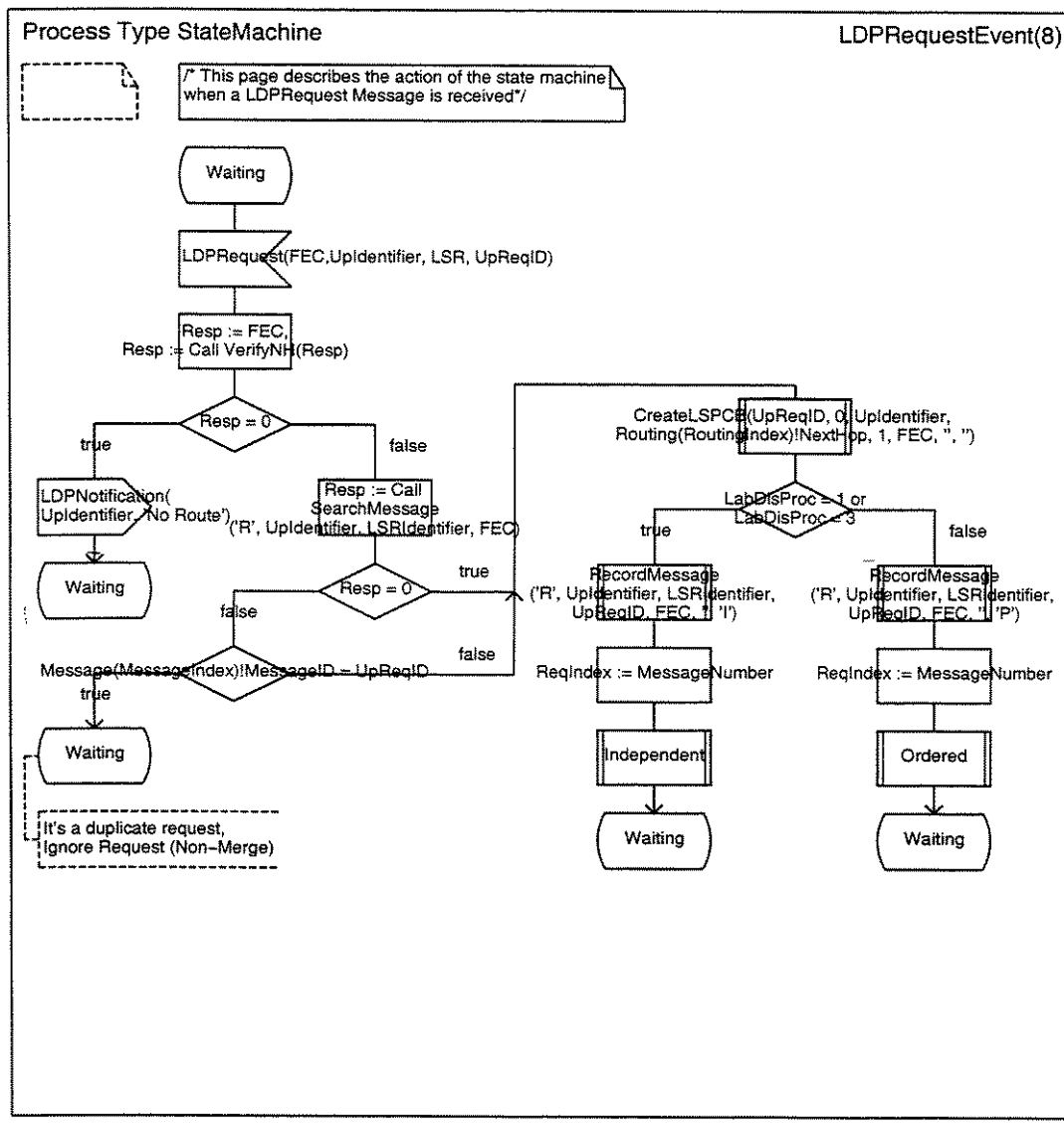

Figura 4.14: Processo *StateMachine* (Recebimento de Mensagem de *Request*)

4.3.2.3 Especificação em SDL - Processo *StateMachine* (Recebimento de Mensagem de *Mapping*)

A figura 4.15 representa o comportamento do LSR diante da ocorrência do recebimento de uma mensagem de mapeamento. De acordo com a figura 4.15, estando o processo *StateMachine* no estado *Waiting*, quando o processo recebe um sinal *LDPMapping*, o processo faz uma chamada ao procedimento *Verify If New NH*. O procedimento *Verify If New NH* irá verificar se o mapeamento recebido, vem de um próximo *hop* novo para uma FEC, e caso venha, atualiza os campos do LSPCB para a FEC.

Em seguida, o processo *StateMachine* faz uma chamada ao procedimento *SearchMessage* para verificar se há uma requisição de label para a FEC, com o identificador de mensagem igual ao identificador da mensagem de mapeamento recebida. Caso haja uma mensagem de requisição de label, é realizada uma chamada ao procedimento *SearchMessage*, para agora verificar se o LSR já havia recebido um mapeamento para a FEC do *peer downstream*. Se o LSR já havia recebido um mapeamento, ele compara se o Label recebido no mapeamento atual, é igual ao label recebido no mapeamento anterior (*Message (MessageIndex)!Label = Label*). Se os labels forem iguais, o LSR envia um sinal *LDPReleaseDown* com a FEC e o label recebido, para o *peer* que enviou a mensagem de mapeamento.

Se não há mensagem de mapeamento do mesmo *peer downstream* previamente recebida, o LSR armazena os dados recebidos no mapeamento fazendo uma chamada ao procedimento *RecordMessage*. Logo após, o LSR fará a distribuição de labels.

Se o LSR estiver configurado para operar no modo *DownStream* (*LabDisProc = 1 ou 2*), o procedimento *DownStream* é ativado. Caso contrário, o procedimento *DownStream On Demand* é ativado.

Retornado ao passo de verificação de mensagem requisição de label (*SearchMessage(R, LSRIdentifier, DownIdentifier, FEC*), se não houve um envio prévio de mensagem de requisição de label para a FEC recebida no mapeamento, o procedimento *VerifyNH* é ativado. O procedimento *VerifyNH* irá verificar se o *peer* que enviou a mensagem de mapeamento, é um próximo *hop* para alguma das FECs reconhecidas pelo LSR. Se o *DownIdentifier*, identificador do *peer downstream* que enviou o mapeamento, não for um próximo *hop* para FECs reconhecidas pelo LSR, os procedimentos de retenção de labels são executados.

Se o LSR estiver configurado para operar no modo de retenção de label conservativo (*LabelRetention = 1*), o procedimento *Conservative* é ativado. Caso contrário, o procedimento *Liberal* é ativado.

4.3.2.4 Especificação em SDL - Processo *StateMachine* (Recebimento de Mensagem de *Release*)

As figuras 4.16 e 4.17 apresentam o comportamento do LSR diante da ocorrência do recebimento de uma mensagem de *release* do LDP. Este evento é representado na especificação, pelo envio do sinal *LDPRelease* do ambiente externo ao sistema MPLS.

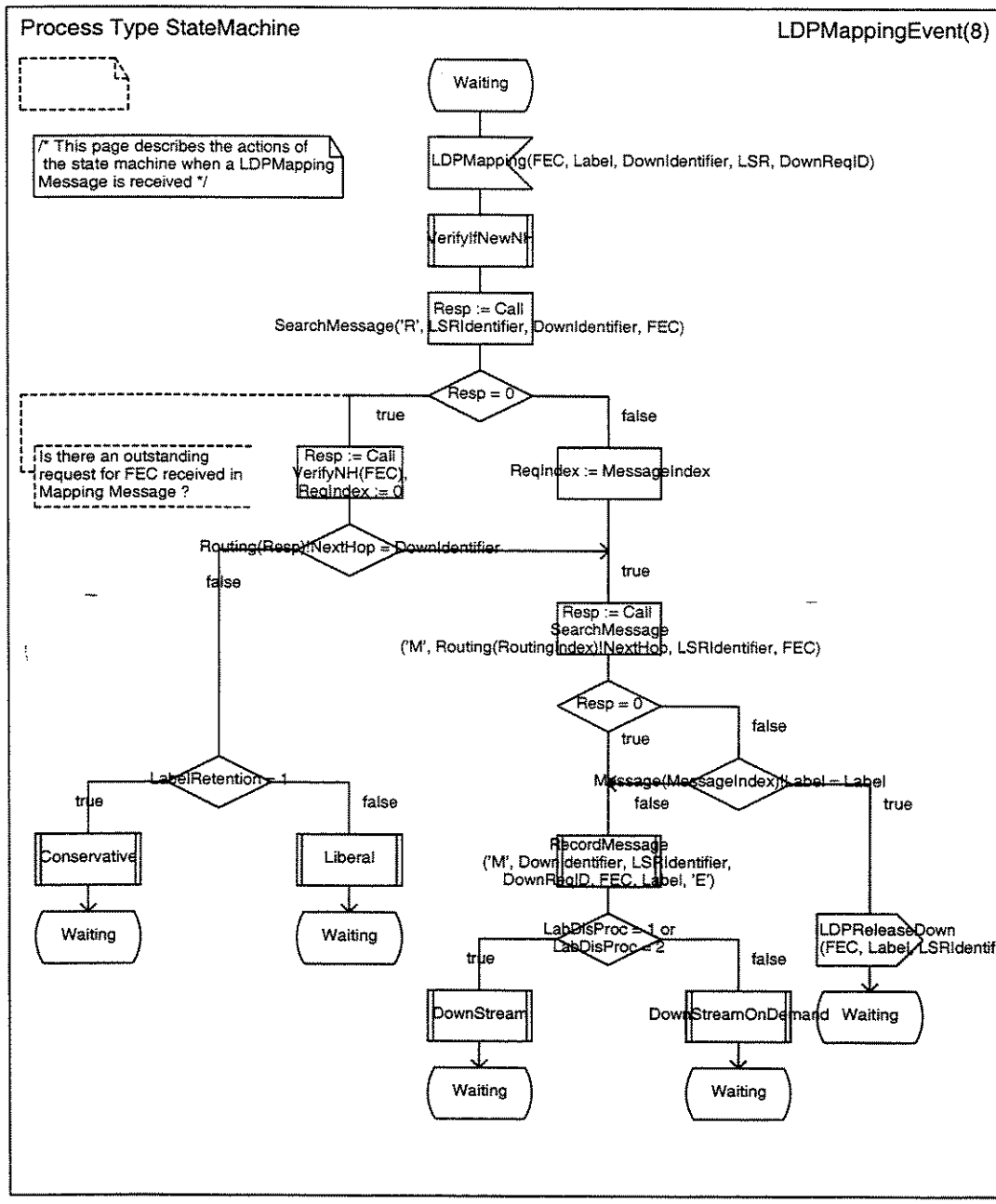

Figura 4.15: Processo *StateMachine* (Recebimento de Mensagem de *Mapping*)

Seguindo-se os quadros do diagrama SDL da figura 4.16 após o recebimento do sinal *LDPRelease*, há a chamada ao procedimento *SearchMessage*.

O procedimento *SearchMessage* irá verificar na tabela de mensagens, se há uma mensagem de *withdraw* pendente para a FEC, previamente enviada pelo LSR ao *peer* que está enviando o *release*. Caso exista a mensagem de *withdraw* pendente, ela é removida da tabela de mensagens com a chamada ao procedimento *DeleteMessage*.

Logo após, o procedimento *VerifyNH* é chamado para verificar se a FEC associada ao label com pedido de *release*, é reconhecida pelo LSR. Caso não seja, o LSR envia uma mensagem de notificação, *LDPNotification*, com a mensagem *No Route*, para o *peer* que enviou o *release*.

Se a FEC for reconhecida pelo LSR, o processo *StateMachine* verifica se o LSR é nó de saída da rede para esta FEC (*Routing(RoutingIndex)!Egress = 1*). Caso ele seja nó de saída da rede, o procedimento *SearchLSP* é chamado para se localizar o índice do LSPCB na tabela de blocos de controle de LSPs.

Encontrado o LSPCB para FEC, o registro de LSPCB é removido através da chamada ao procedimento *DeleteLSPCB* e, o label *upstream* especificado na mensagem de *release*, é liberado através da chamada ao procedimento *FreeLabel*.

Se o LSR não for nó de saída da rede para a FEC, é necessário verificar se a mensagem de *release* precisa ser propagada para os *peers downstream*.

O comportamento do LSR diante do evento de recebimento de uma mensagem de *release*, continua na figura 4.17 à partir do ponto *Continued*.

Após o símbolo *Continued*, o procedimento *SearchLSP* é executado para se encontrar o registro LSP associado à FEC na tabela de LSPCBs. Se o label *downstream* for diferente de branco, ou seja, nulo, o processo *StateMachine* tomará as seguintes ações : se o label *upstream* já foi alocado (*LSPCB (LSPCBIndex) !UpLabel = nulo*), libera-se o label que foi especificado na mensagem *release*, através da chamada ao procedimento *FreeLabel*.

Se o LSR estiver operando no modo de controle de LSP Ordenado, isto é, *LasDisProc diferente de 1 e 3*, o label *upstream* não foi alocado, já que o label *downstream* é nulo e, o LSR não é nó de saída da rede para a FEC. Esta conclusão vem do fato de que, no modo ordenado, um label só é propagado para o *peer upstream* se o LSR for nó de saída da rede para a FEC, ou se já tiver recebido um mapeamento do próximo *hop* para a FEC.

Desta forma, há uma requisição de label pendente para a FEC, que é encontrado através da chamada do procedimento *SearchMessage*. A mensagem de requisição de label pendente para a FEC é removida através do procedimento *DeleteMessage*.

Então, é verificado se há uma mensagem de requisição de label pendente do LSR para o próximo *hop* para FEC, executando-se mais uma vez o procedimento *SearchMessage*. Se a mensagem for encontrada, é removida da tabela de mensagens através da chamada ao procedimento *DeleteMessage*. Faz-se a chamada ao procedimento *DeletePreviousMapping*, para que as mensagens de mapeamento para a FEC, se existirem, sejam removidas da tabela de mensagens. Logo após, o LSPCB para a FEC é removido através da chamada ao procedimento *DeleteLSPCB*.

Agora, se o label *downstream* já tiver sido alocado (*LSPCB(LSPCBIndex)!DownLabel diferente de nulo*), como consequência, o label *upstream* já foi alocado, mesmo se o LSR es-

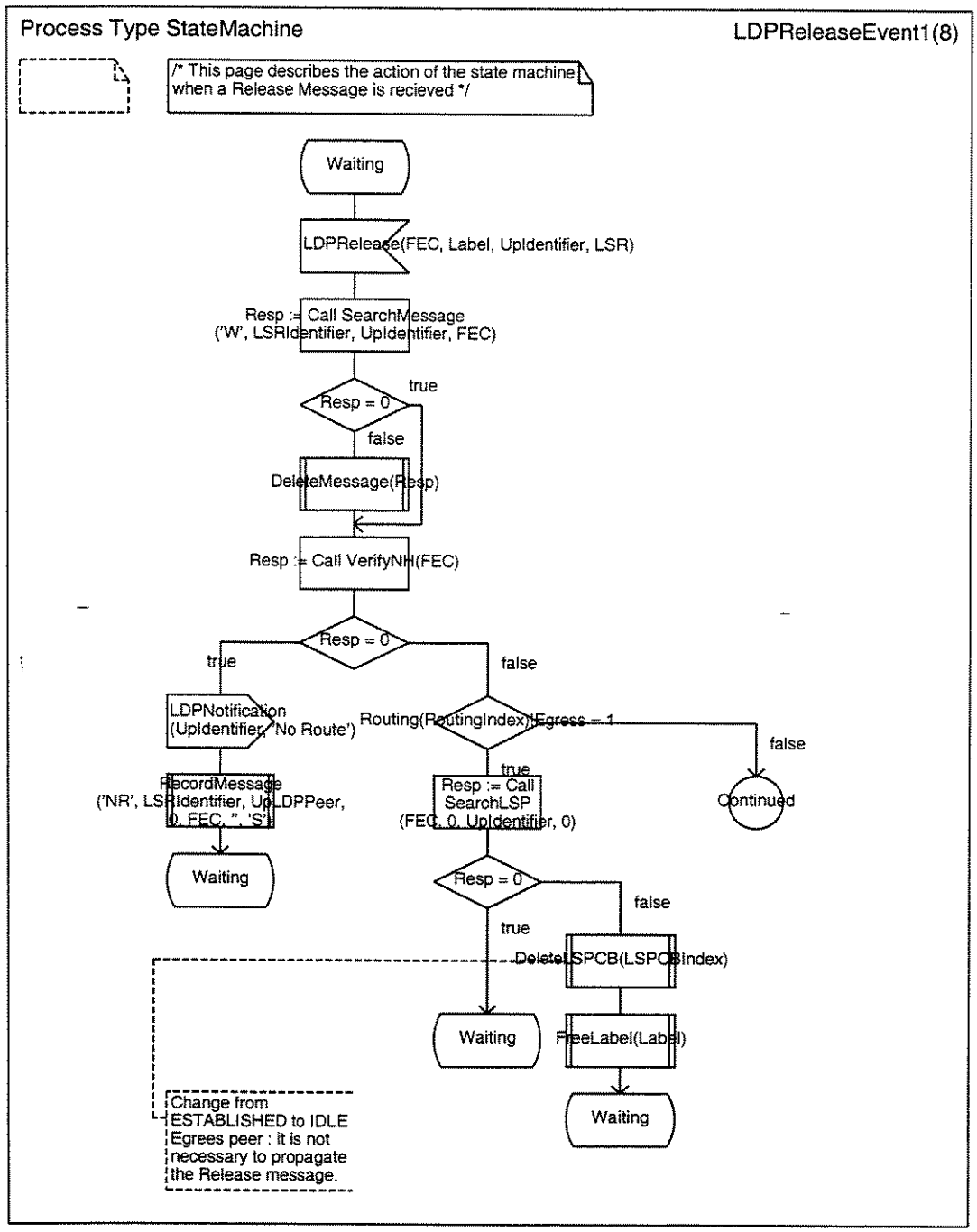

Figura 4.16: Processo *StateMachine* (Recebimento de Mensagem de Release)

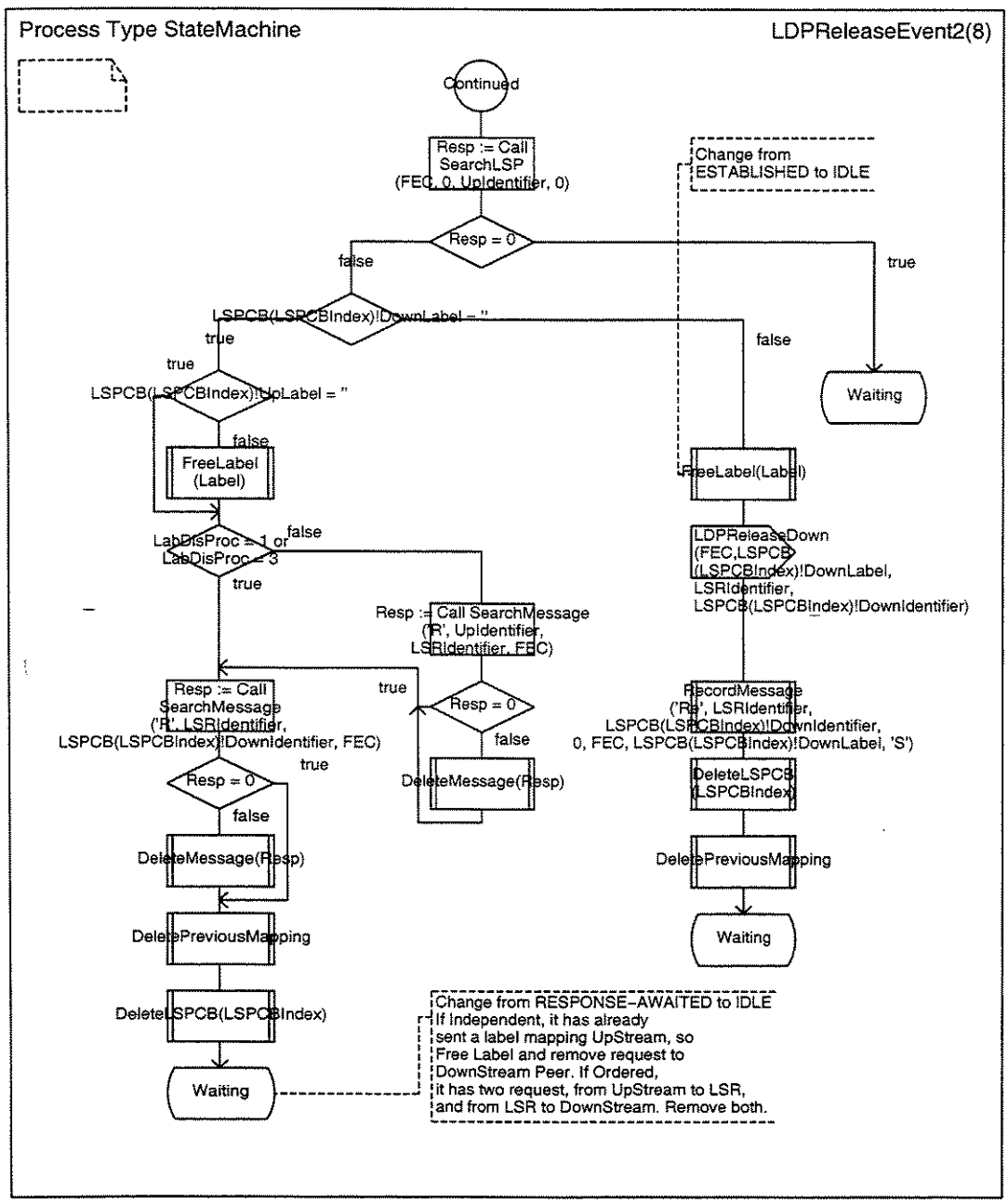

Figura 4.17: Processo *StateMachine* (**Continuação**) Recebimento de Mensagem de Release)

tiver operando no modo independente, quanto no modo ordenado e, não há mensagens de requisição de label pendentes. Desta forma, o processo *StateMachine* propaga a mensagem de *release* para o próximo *hop* para a FEC através do envio do sinal *LDPReleaseDown*.

O envio da mensagem de *release* é registrado na tabela de mensagens com a chamada ao procedimento *RecordMessage* e, o LSPCB é removido da tabela de LSPCBs com a chamada ao procedimento *DeleteLSPCB*.

4.3.2.5 Especificação em SDL - Recebimento de Mensagem *WithDraw* (Processo *StateMachine*)

A figura 4.18 apresenta o comportamento do processo *StateMachine* diante da ocorrência do recebimento de uma mensagem de *WithDraw* do LDP.

Acompanhando-se os símbolos SDL da figura 4.18, estando o processo *StateMachine* no estado *Waiting*, após o recebimento de um sinal *LDPWithDraw*, o LSR responde ao *peer downstream* que enviou a mensagem de *withdraw*, com uma mensagem de *release* representada pelo envio do sinal *LDPReleaseDown*.

O sinal *LDPRelaseDown* é enviado com os parâmetros FEC e Label enviados na mensagem de *withdraw*, identificador do LSR e, o identificador do *peer downstream*, também recebido na mensagem *withdraw*.

O passo seguinte é registrar o envio da mensagem de *release* através da chamada ao procedimento *RecordMessage*. Em seguida, faz-se uma chamada ao procedimento *SearchMessage*, para a realização de uma consulta na tabela de mensagens, para verificar se, o LSR recebeu anteriormente uma mensagem de mapeamento para a FEC. Caso o LSR não tenha recebido mapeamento do *peer downstream* que enviou a mensagem de *withdraw* para a FEC, o processo retorna ao estado *Waiting* e, fica aguardando pelo próximo evento.

Caso exista a mensagem de mapeamento, ela é removida da tabela de mensagens através da chamada ao procedimento *DeleteMessage*. O LSPCB para a FEC é então procurado na tabela de LSPCBs através da chamada do procedimento *SearchLSP*. Se há um LSPCB para a FEC, de identificador *downstream* igual ao identificador do *peer* que enviou o *withdraw*, o label *downstream*, isto é, o label pelo qual o *peer downstream* está pedindo o *withdraw*, é removido do bloco de controle de LSP para a FEC.

O processo *StateMachine* vai então verificar, se ele propagou *mappings* para esta FEC aos seus LDP *Peers UpStream*, através de uma consulta a tabela de mensagens, chamando o procedimento *SearchMessage*.

Se houve a propagação de mapeamento para a FEC ao LDP *Peer UpStream*, a mensagem de *withdraw* é propagada para o *peer upstream* representada pelo envio do sinal *LDPWithDrawUp*. Os parâmetros do sinal *LDPWithDrawUp* são a FEC recebida na mensagem *withdraw*, o label *upstream* do LSPCB para a FEC (*LSPCB(LSPCBIndex)!UpLabel*), o identificador do LSR, e o identificador do *peer UpStream* para a FEC (*LSPCB LSPCB*) *!UpIdentifier*).

O último passo é registrar o envio da mensagem de *withdraw* propagada *upstream*, através da chamada ao procedimento *RecordMessage*. O processo *StateMachine* volta ao estado *Waiting* pronto para a ocorrência do próximo evento.

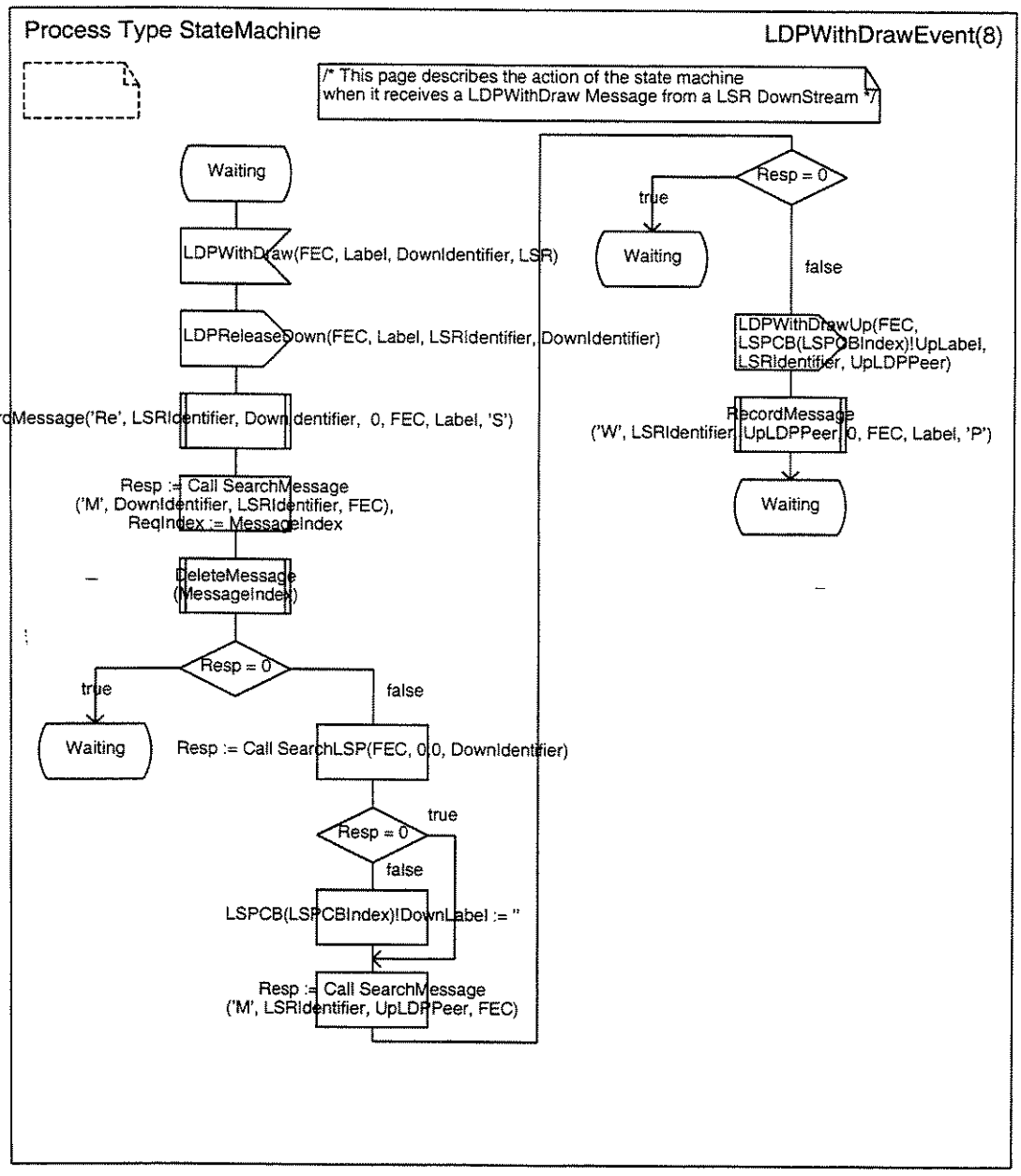

Figura 4.18: Processo *StateMachine* (Recebimento de uma Mensagem de *WithDraw*)

4.3.2.6 Especificação em SDL - Processo *StateMachine* (Detecção de um próximo *hop* novo para uma FEC)

O evento de detecção interna de um próximo *hop* novo para uma FEC pelo LSR, é representado nesta especificação, pelo envio do sinal *IntNewNH*, como mostrado na figura 4.19.

Os parâmetros passados no sinal *IntNewNH* são a FEC, o próximo *hop* novo para a FEC, o antigo próximo *hop* para a FEC, e o antigo label, se houver, enviado pelo antigo próximo *hop* para o LSR.

Como pode-se observar a sequência de símbolos SDL apresentada na figura 4.19, estando o processo *StateMachine* no estado *Waiting*, após o recebimento do sinal *IntNewNH*, a tabela de roteamento do LSR é atualizada através da chamada ao procedimento *Update Routing Table*.

Após a atualização da tabela de roteamento, o LSR verifica se havia recebido um mapeamento de label do antigo próximo *hop* para a FEC chamando o procedimento *SearchMessage*. Caso tenha recebido, a mensagem de mapeamento é removida da tabela de mensagens, através da chamada do procedimento *DeleteMessage*.

A próxima ação é verificar se o LSR está configurado para operar com modo de retenção de label Liberal (*LabelRetention = 2*). Caso ele esteja operando no modo Liberal, ele pode em algum instante anterior à detecção de mudança de próximo *hop*, ter recebido e retido um mapeamento de label para a FEC do próximo *hop* novo. É verificado se o LSR recebeu anteriormente um mapeamento de label para a FEC do próximo *hop* novo, através da chamada do procedimento *SearchMessage*.

Se o LSR não recebeu a mensagem de mapeamento do próximo *hop* novo, ele realiza as mesmas ações que ele faria, se estivesse configurado para operar com modo de retenção de label Conservativo, ou se o novo Next-Hop estivesse operando com procedimento de distribuição *DownStream-On-Demand*.

É importante salientar que, nesta especificação não é possível configurar o procedimento de distribuição de labels dos LDP *Peers* ao LSR. O procedimento dos LDP *Peers* é observado de acordo com as mensagens que são enviadas do ambiente ao sistema. Isto significa que, os sinais enviados do ambiente ao sistema durante as simulações, especificarão o tipo de distribuição dos LDP *Peers*.

As ações tomadas pelo LSR são então, na sequência apresentada no diagrama da figura 4.19, primeiramente enviar uma mensagem de *release* para o antigo próximo *hop*, através do envio da mensagem *LDPReleaseDown*. Logo após, localizar o LSPCB para a FEC através da chamada ao procedimento *SearchLSP*.

Em seguida, alguns campos do registro do LSP para a FEC são alterados. O label *downstream* é reinicializado para nulo (*DownLabel*), o campo do identificador do próximo *hop* novo, *NewNHIdentifier*, recebe o valor do próximo *hop* novo recebido no sinal *IntNewNH*, o identificador da mensagem de requisição de label para o próximo *hop* novo recebe o valor da variável *GenerateID* e, o estado do LSP muda de *Established* para *New NH Response Awaited*.

Um pedido de requisição de label é enviado para o próximo *hop* novo através do envio

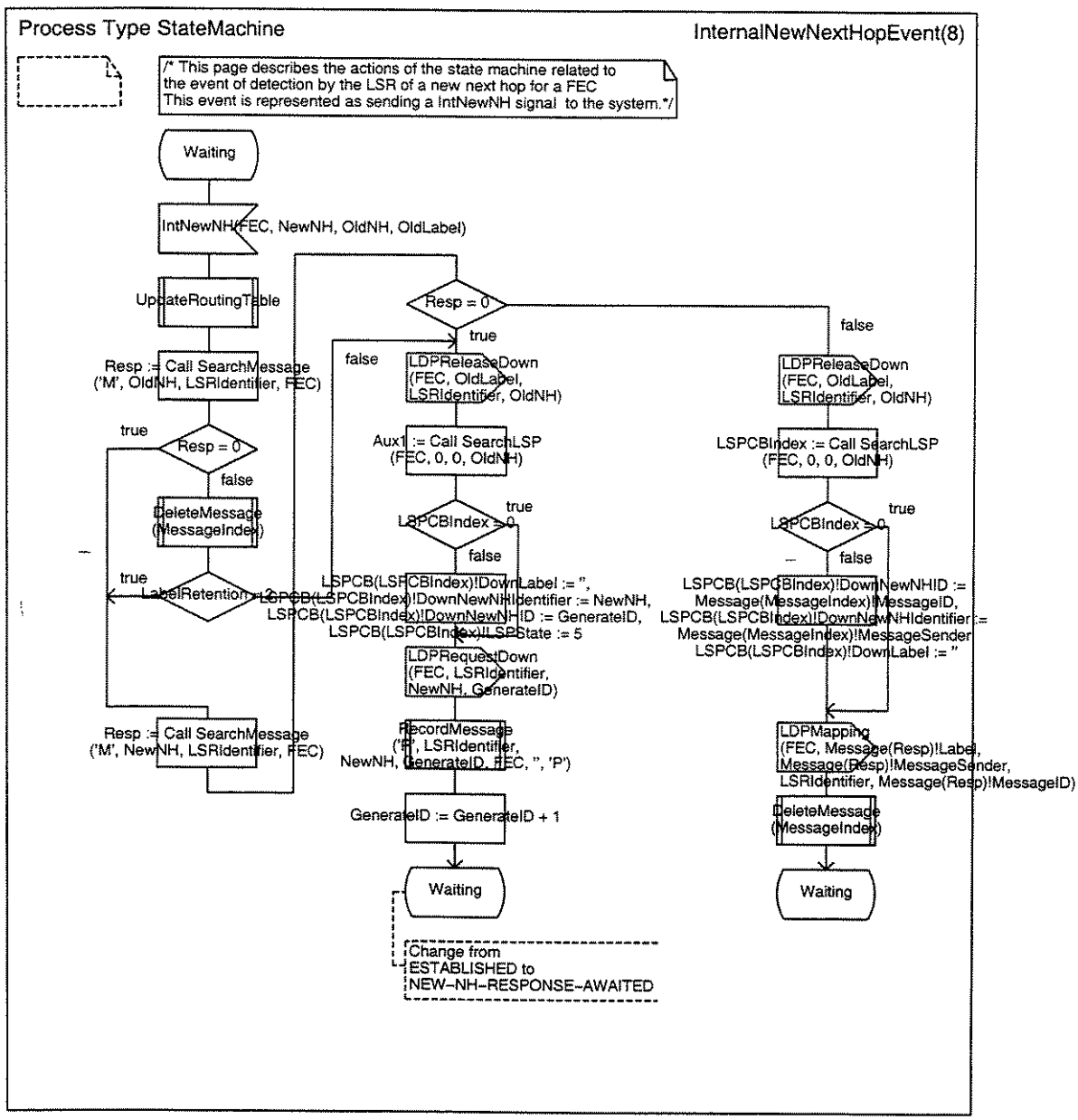

Figura 4.19: Processo *StateMachine* (Detecção de um próximo *hop* novo para uma FEC)

do sinal *LDPRequestDown*. Os parâmetros do sinal são : a FEC que recebeu mudança de próximo *hop*, o identificador do LSR, o identificador do próximo *hop* e, o valor do identificador da mensagem de requisição de label, *GenerateID*. O envio da mensagem é registrada na tabela de mensagens através da chamada do procedimento *RecordMessage*.

O valor da variável que gera identificadores únicos para as mensagens enviadas pelo LSR, *GenerateID*, é incrementado de uma unidade e, o processo retorna ao estado *Waiting* pronto para o próximo evento.

Se o LSR já tiver recebido uma mensagem do próximo *hop* novo para a FEC, não será necessário enviar uma requisição de label. O LSR poderá executar a operação de *LOCAL REPAIR* como mostrado no diagrama SDL da figura 4.19, a sequência de símbolos mais à direita na figura.

Primeiramente, o LSR envia uma mensagem de *release* para o antigo próximo *hop*, através do envio do sinal *LDPReleaseDown*. Localiza-se o LSPCB para a FEC na tabela de LSPCBs, através do procedimento *SearchLSP*.

Atualiza-se os campos do LSPCB da seguinte maneira : o identificador de mensagem de requisição de label para o próximo *hop* novo, recebe o valor do identificador da mensagem de mapeamento enviada pelo próximo *hop* novo. O campo de label *downstream* é atualizado com o label recebido na mensagem de mapeamento e então, a máquina de estado deve comportar-se como se estivesse recebendo o sinal de mapeamento neste instante.

Para isso é enviado o sinal de mapeamento ao processo *StateMachine*, que ao ser consumido (ações descritas no diagrama que representa o recebimento de uma mensagem de mapeamento), atualizará os campos do LSPCB para a FEC e, o LSPCB poderá retornar ao estado *Established*.

A mensagem de mapeamento previamente recebida pelo próximo *hop* novo é removida da tabela de mensagens (procedimento *DeleteMessage*) e, o processo retorna ao estado *Waiting* pronto para o próximo evento.

4.4 Resultados das Simulações da Máquina de Estado e dos Procedimentos de Distribuição de Label do LDP

Esta seção apresenta os resultados das simulações da especificação do sistema MPLS apresentado na seção 4.3. O objetivo das simulações é verificar se o LSR especificado comporta-se adequadamente em relação ao recebimento de mensagens LDP, de acordo com os procedimentos descritos em [21]. Mudando-se a configuração do LSR, verifica-se como o LSR manipula as tabelas de roteamento, de mensagens, de controle de LSP e de labels, além do envio de sinais para os seus LDP *peers*. O objetivo das simulações é de também verificar as máquinas de estado do LDP da seção 4.2 que, são geradas durantes as simulações, conforme os LSPs vão sendo estabelecidos.

As subseções 4.4.1, 4.4.2, 4.4.3, 4.4.4 e 4.4.5 apresentam as simulações de um LSR

configurado para operar em diversas configurações. A subseção 4.4.1 apresenta uma simulação com uma sequência de eventos maior. As outras subseções apresentam pequenos exemplos sobre o comportamento que é relevante à configuração do LSR.

As simulações, consistem no envio de uma sequência de sinais do ambiente externo ao sistema MPLS e, na observação da resposta do LSR diante da ocorrência dos eventos de recebimento destes sinais. Os resultados são apresentados através de diagramas de MSC nas figuras 4.20, 4.21, 4.22, 4.23 e 4.24.

Os diagrama MSC representam a troca de sinais entre o ambiente externo, primeira linha vertical e, entre o processo *StateMachine*, subseção 4.3.2, na segunda linha vertical. O envio de sinais do ambiente externo ao processo *StateMachine*, representa a ocorrência de eventos de recebimento de mensagens LDP.

Entretanto, como pretende-se observar a máquina de estado LDP dos LSPs a serem estabelecidos e, como o estado do LSP é armazenado em um campo da estrutura da tabela de LSPs (*LSPState*), seção 4.3, além dos MSCs, é preciso observar os valores das principais variáveis e estruturas do sistema.

Por este motivo, o apêndice A apresenta uma descrição mais detalhada da saída do *Simulator* para o sistema MPLS, isto é, uma descrição do formato que é apresentado uma consulta aos valores das variáveis do sistema MPLS no *Simulator*.

O apêndice B apresenta uma descrição mais detalhada da simulação da subseção 4.4.1, isto é, apresenta os valores das variáveis do sistema MPLS descritas no apêndice A, após a ocorrência de cada um dos eventos, além da descrição do diagrama MSC. Através da descrição dos valores dos campos da estrutura de LSPs, pode-se observar a mudança de estados da máquina do LDP para cada um dos LSPs a ser estabelecidos durante as simulacções.

Entretanto, somente as variáveis relevantes a cada caso simulado serão apresentadas. Isto significa que, somente as variáveis que sofrerem alterações após o recebimento de um sinal (representando a ocorrência de um evento), serão apresentadas. Desta forma, torna-se possível comparar valores de variáveis antes e após as transições realizadas.

Nas simulações apresentadas neste capítulo, assim como a simulação do apêndice B, os valores atribuídos às variáveis que representam endereços IP, FECs ou labels, são valores fictícios, ou seja, valores simbólicos utilizados para facilitar a compreensão das simulações. As simulações são realizadas no *Simulator* do SDT e, consistem de uma sequência de envio de sinais do ambiente externo ao sistema MPLS.

4.4.1 Simulação de um LSR operando com Modo de Controle LSP Independente

Esta subseção apresenta a simulação de um LSR configurado para operar no modo de controle LSP Independente, subseção 4.2.1. A simulação é apresentada no diagrama MSC (representando os sinais enviados do ambiente externo ao sistema e, do sistema ao ambiente externo), da figura 4.20.

De acordo com a sequência da ocorrência de eventos da figura 4.20, o processo encontra-se primeiramente no estado *Waiting*. O estado *Waiting* indica que o processo está pronto

para receber sinais do ambiente externo.

Envio do sinal *LDPRequest 1 123 1111 11* :
Logo a seguir, o ambiente externo envia um sinal *LDPRequest* ao processo *StateMachone* com os parâmetros (1=FEC, 123=identificador do *peer* que enviou a mensagem (*Peer UpStream*), 1111=identificador do *peer* que recebeu a mensagem (LSR) e, 11=identificador da mensagem). O envio deste sinal representa o envio de uma mensagem LDP de requisição de label, enviada de um *Peer UpStream* ao LSR especificado.

Como o LSR é nó de saída da rede para a FEC 1 e está operando no modo independente, ele aloca um label *upstream* e, envia uma mensagem de mapeamento ao *peer* que requisitou o label. O LSR não propaga o *request downstream* porque ele é nó de saída da rede para a FEC. O LSP para a FEC 1 vai para o estado *Established.*

O envio da mensagem de mapeamento é representado pelo envio do sinal *LDPMapping* com os parâmetros (1=FEC, A=label *upstream* alocado, 1111=identificador do *peer* que enviou a mensagem (LSR), 123=identificador do *peer* que recebeu a mensagem (*Peer UpStream*), 11=identificador da mensagem de requisição de label ao qual o mapeamento está respondendo e, *NOT PROPAGATING* porque o LSR não está propagando um mapeamento enviado de um *peer downstream.* O processo *StateMachine* volta ao estado *Waiting* aguardando pelo envio do próximo sinal.

Envio do sinal *LDPRequest 2 123 1111 12* :
O próximo sinal enviado do ambiente externo ao processo *StateMachine* é o sinal *LDPRequest*, figura 4.20, com os parâmetros (2=FEC, 123=*peer* que enviou a mensagem, 1111=*peer que recebeu a mensagem* e, 12=identificador da mensagem).

Como o LSR está operando no modo independente, ele aloca um label *upstream* e envia um mapeamento para o *peer upstream* requisitante. O envio do mapeamento é representado, na figura 4.20, pelo envio do sinal *LDPMapping* do processo *StateMachine* ao ambiente externo com os parâmetros (2=FEC, B=label *upstream* alocado, 1111=*peer* que enviou a mensagem (LSR), 123=*peer* que recebeu a mensagem (*Peer UpStream*), 12=identificador da mensagem de requisição de label ao qual o mapeamento está respondendo e, *NOT PROPAGATING* porque o LSR não está propagando um mapeamento enviado de um *peer downstream.*

Entretanto, como o LSR não é nó de saída da rede para a FEC 2, ele propaga a mensagem de *request downstream.* A propagação da mensagem de requisição de label é representada, na figura 4.20, pelo envio do sinal *LDPRequestDown* do processo *StateMachine* ao ambiente externo com os parâmetros (2=FEC, 1111=*peer* que enviou a mensagem (LSR), 2222=*peer* que recebeu a mensagem (próximo *hop* para a FEC) e, 800=identificador da mensagem). O LSP para a FEC 2 vai para o estado *Response Awaited* aguardando pelo envio de um mapeamento do próximo *hop.* O processo retorna ao estado *Waiting* aguardando pelo envio do próximo sinal.

Envio do sinal *LDPMapping 2 X 2222 1111 800* :
O próximo sinal enviado do ambiente externo ao processo *StateMachine*, é o sinal

LDPMapping com os parâmetros (2=FEC, X=label *downstream*, 2222=*peer* que enviou a mensagem (próximo *hop* para a FEC), 1111=*peer* que recebeu a mensagem (LSR) e, 800=identificador da mensagem de requisição de label ao qual o mapeamento está respondendo).

O LSR processa as informações recebidas no mapeamento em resposta ao pedido de requisição de label que estava pendente. Como o LSR já havia propagado um label *upstream* quando recebeu uma requisição de label para a FEC 2, ele não propaga o *mapping upstream*. Ele associa o label *downstrea* recebido na mensagem de mapeamento, label X, ao label *upstream* previamente alocado, label B. O LSP para a FEC 2 passa do estado *Response Awaited* para *Established*. O processo *StateMachine* retorna ao estado *Waiting* aguardando pelo envio do próximo sinal.

Envio do sinal *IntNewNH 2 4444 2222 X* :

O próximo sinal enviado, figura 4.20, é o sinal *IntNewNH* com os parâmetros (2=FEC, 4444=próximo *hop* novo para a FEC, 2222=Antigo próximo *hop* para a FEC, X=Antigo label *downstream*). O envio deste sinal do ambiente extern para o process *StateMachine* representa que, o LSR detectou uma mudança de próximo *hop* para a FEC 2. Desta forma, o LSR atualiza a sua tabela de roteamente substituindo o próximo *hop* para a FEC 2 de 2222 para 4444 e envia uma mensagem de *release* para o antigo próximo *hop*.

O envio da mensagem de *release* para o antigo próximo *hop* é representado, figura 4.20, pelo envio do sinal *LDPReleaseDown* do processo *StateMachine* para o ambiente com os parâmetros (2=FEC, X=label *downstream*, 1111=identificador do LSR, 2222=antigo próximo *hop*).

Além disso, o LSR envia uma mensagem de requisição de label para o próximo *hop* novo que, é representado na figura 4.20, pelo envio do sinal *LDPRequestDown* do processo *StateMachine* ao ambiente com os parâmetros (2=FEC, 1111=*peer* que enviou a mensagem (LSR), 4444=*peer* que recebeu a mensagem (próximo *hop* novo) e, 801=identificador da mensagem). O LSP para a FEC 2 passa do estado *Established* para o estado *New NH Response Awaited*. O LSP permanece no estado *New NH Response Awaited* até que receba um mapeamento do próximo *hop* novo para a FEC. O processo *StateMachine* retorna ao estado *Waiting* aguardando pelo envio do próximo sinal.

Envio do sinal *LDPMapping 2 Y 4444 1111 801* :

O próximo sinal enviado, figura 4.20, é o sinal *LDPMapping* com os parâmetros (2=FEC, Y=label *upstream*, 4444=*peer* que enviou a mensagem (próximo *hop* para a FEC), 1111=*peer* que receberá a mensagem (LSR) e, 801=identificador da mensagem de requisição de label ao qual o mapeamento está respondendo).

Ao receber o mapeamento do próximo *hop* novo, o LSR realiza o procedimento de *LOCAL REPAIR*, isto é, associa o label *downstream* recebido na mensagem de mapeamento do próximo *hop* novo, label Y, ao antigo label *upstream* para a FEC 2, label B. O LSP para a FEC 2, passa do estado *New NH Response Awaited* para o estado *Established*. O processo *StateMachine* retorna ao estado *Waiting* aguardando pelo envio do próximo sinal.

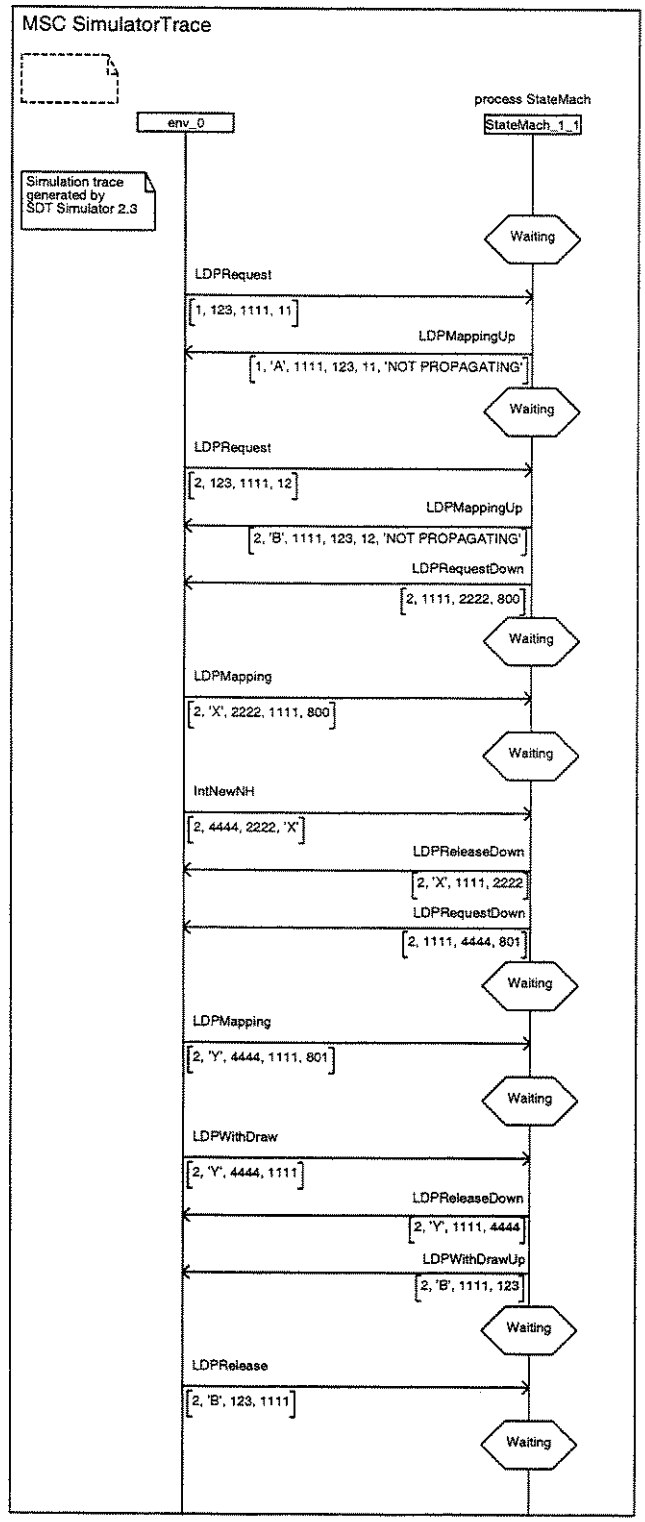

Figura 4.20: MSC Simulação de um LSR com Modo de Controle LSP Independente

Envio do sinal *LDPWithDraw 2 Y 4444 1111* :
O próximo sinal enviado do ambiente externo para o processo *StateMachine*, figura 4.20, é o sinal *LDPWithDraw* com os parâmetros (2=FEC, Y=label *upstream*, 4444=*peer* que enviou a mensagem (próximo *hop* para a FEC e, 1111=*peer* que recebeu a mensagem (LSR)).

Ao receber uma mensagem de *withdraw*, o LSR responde ao *peer* requisitante com uma mensagem de *release*, dizendo que o label *downstream* foi liberado. O envio da mensagem de *release* é representado na figura 4.20 pelo envio do sinal *LDPRelease* do processo *StateMachine* ao ambiente com os parâmetros (2=FEC, Y=label *upstream* liberado, 1111=*peer* que enviou a mensagem (LSR) e, 4444=*peer* que recebeu a mensagem (próximo *hop* que enviou o *withdraw*)).

Além disso, o LSR propaga a mensagem de *withdraw* para o *Peer UpStream*. A propagação da mensagem de *withdraw* é representada na figura 4.20 pelo envio do sinal *LDPWithDrawUp* do processo *StateMachine* ao ambiente com os parâmetros (2=FEC, B=label *downstream*, 1111=*peer* que enviou a mensagem (LSR) e, 123=*peer* que recebeu a mensagem (*Peer Upstream* para a FEC)). O processo *StateMachine* retorna ao estado *Waiting* aguardando pelo envio do próximo sinal.

Envio do sinal *LDPRelease 2 B 123 1111* :
O próximo sinal enviado do ambiente ao processo *StateMachine*, figura 4.20, é o sinal *LDPRelease* com os parâmetros (2=FEC, B=label *upstream*, 123=*peer* que enviou a mensagem (*Peer UpStream*) e, 1111=*peer* que recebeu a mensagem (LSR)).

O envio deste sinal representa a resposta do LDP *Peer UpStream*, à mensagem de *withdraw* enviada ao *peer* pelo LSR. Com o recebimento da mensagem de *release*, o LSR considera que o label *upstream* B foi liberado no *Peer UpStream* e então, o LSR destrói o registro que armazenava informações sobre o LSP para a FEC 2. Desta forma, o LSP para a FEC 2 passa do estado *Established* para o estado *Idle*. O processo *StateMachine* retorna ao estado *Waiting* aguardando pelo envio do próximo sinal.

4.4.2 Simulação de um LSR operando com Modo de Controle LSP Ordenado

Esta subseção apresenta a simulação do sistema MPLS, onde o LSR é configurado para operar no modo de controle LSP Ordenado, subseção 4.2.2. O objetivo desta simulação é de mostrar o comportamento do LSR com controle LSP Ordenado, diante do estabelecimento de um LSP. O exemplo de simulação da figura 4.21 apresenta somente a parte relevante ao modo de operação Ordenado, ou seja, a parte que difere da operação no modo Independente.

De acordo com a figura 4.21, o processo *StateMachine* encontra-se inicialmente no estado *Waiting* pronto para receber sinais do ambiente externo.

Envio do sinal *LDPRequest 2 123 1111 10* : O primeiro sinal enviado do ambiente externo para o processo *StateMachine*, é o sinal LDPRequest com os parâmetros (2=FEC,

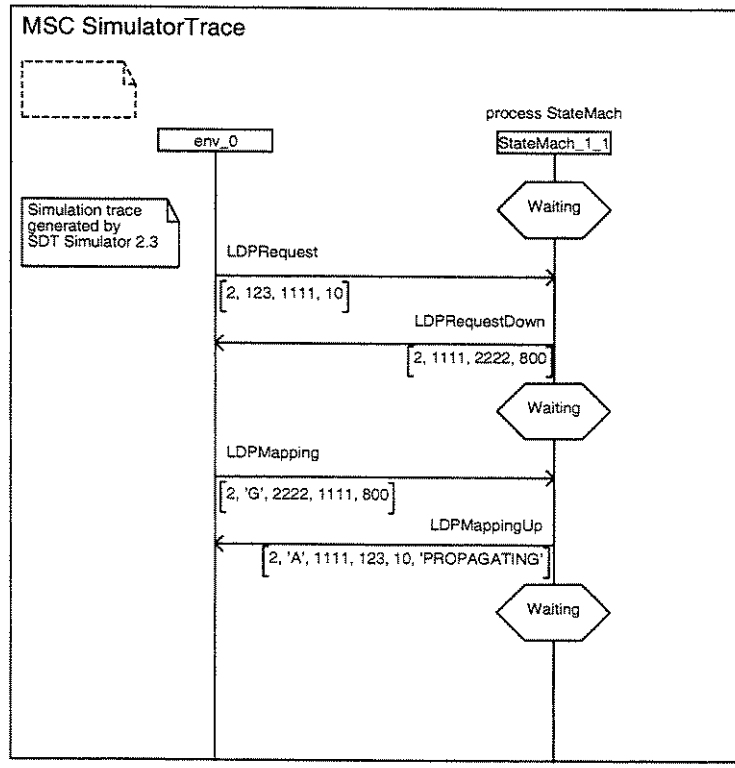

Figura 4.21: MSC Simulação de um LSR com Modo de Controle LSP Ordenado

123=*peer* que enviou a mensagem (*Peer UpStream*), 1111=*peer* que recebeu a mensagem (LSR) e, 10=identificador da mensagem).

Como o LSR está configurado para operar no modo de controle Ordenado, ao receber uma mensagem de requisição de label para uma FEC, o LSR não aloca um label *upstream* e responde com um mapeamento ao *Peer UpStream*. Operando no modo de controle Ordenado, o LSR só envia um mapeamento se ele for nó de saída da rede para a FEC, ou se ele já tiver recebido um mapeamento do próximo *hop* para a FEC.

Desta forma, o LSR propaga a mensagem de *request downstream* e aguarda por um mapeamento do próximo *hop*. A propagação da mensagem de mapeamento é representada na figura 4.21, pelo envio do sinal *LDPRequestDown* do processo *StateMachine* ao ambiente com os parâmetros (2=FEC, 1111=*peer* que enviou a mensagem (LSR), 2222=*peer* que recebeu a mensagem (próximo *hop* para a FEC) e, 800=identificador da mensagem). O LSP para a FEC 2 vai do estado *Idle* para *Response Awaited*, permanecendo neste estado até que receba um mapeamento do próximo *hop*. O processo *StateMachine* vai para o estado *Waiting* pronto para receber o próximo sinal.

Envio do sinal *LDPMapping 2 G 2222 1111 800* :

O próximo sinal enviado do ambiente ao processo *StateMachine*, é o sinal *LDPMapping* com os parâmetros (2=FEC, G=label *downstream*, 2222=*peer* que enviou a mensagem (próximo *hop* para a FEC), 1111=*peer* que receberá a mensagem (LSR) e, 800=identifi-

cador da mensagem de requisição de label ao qual o mapeamento está respondendo).

Ao receber o mapeamento do próximo *hop* para a FEC, o LSR aloca um label *upstream*, associa o label *downstream* recebido no mapeamento, label G, ao label *upstream* alocado, label A, e envia uma mensagem de mapeamento ao *Peer UpStream*, atendendo a mensagem de requisição de label que estava pendente.

O envio da mensagem de mapeamento é representado na figura 4.21, pelo envio do sinal *LDPMappingUp* do processo *StateMachine* ao ambiente com os parâmetros (2=FEC, A=label *upstream* alocado, 1111=peer que enviou a mensagem (LSR), 123=*peer* que receberá a mensagem (*Peer UpStream*), 12=identificador da mensagem de requisição de label ao qual o mapeamento está respondendo e, *Propagating* porque o LSR está propagando um mapeamento recebido de um *peer downstream*). O LSP passa do estado *Response Awaited* para *Established* e, o processo *StateMachine* volta ao estado *Waiting* pronto para receber o próximo sinal.

4.4.3 Simulação de um LSR operando no Modo de Distribuição de Label DownStream

Esta subseção apresenta a simulação de um LSR operando no modo de distribuição de label *DownStream*, subseção 4.2.3. O exemplo de simulação apresentado no diagrama MSC da figura 4.22, tem por objetivo observar o comportamento do LSR, diante do recebimento de uma mensagem de mapeamento de um *Peer DownStream*, para uma FEC que não foi requisitada por um *Peer UpStream*.

De acordo com a figura 4.22, o processo *StateMachine* encontra-se inicialmente no estado *Waiting* pronto para receber sinais do ambiente externo.

Envio do sinal *LDPMapping 2 D 2222 1111 900* :

O primeiro sinal enviado do ambiente ao processo *StateMachine*, é o sinal *LDPMapping* com os parâmetros (2=FEC, 2222=*peer* que enviou a mensagem (próximo *hop* para a FEC), 1111=*peer* que receberá a mensagem (LSR) e, 900=identificador da mensagem).

Ao receber um mapeamento de um *peer downstream*, o LSR verifica se há um pedido de requisição de label pendente para a FEC. Neste caso, não há um pedido pendente. Entretanto, como o LSR está configurado para operar no modo *DownStream*, ele aloca um label *upstream*, label A, associa o label *upstream alocado* ao label *downstream* recebido no mapeamento, label D e, propaga o *mapping upstream*.

A propagação do *mapping upstream* é representado na figura 4.22, pelo envio do sinal *LDPMappingUp* com os parâmetros (2=FEC,A=label *upstream* alocado, 1111=*peer* que enviou a mensagem (LSR), 123=*peer* que receberá a mensagem (*Peer UpStream* para a FEC), 800=identificador da mensagem e, *PROPAGATING* porque o LSR está propagando um mapeamento recebido de um *peer downstream*). O LSP passa do estado *Idle* para o estado *Established*. O processo *StateMachine* volta ao estado *Waiting* pronto para receber sinais do ambiente externo.

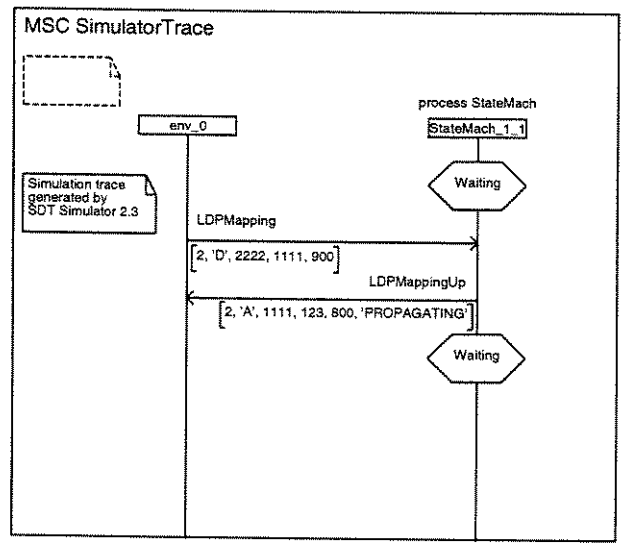

Figura 4.22: MSC Simulação de um LSR com Modo de Distribuição de Label *DownStream*

4.4.4 Simulação de um LSR operando no Modo de Distribuição de Label DownStream On Demand

Esta subseção apresenta a simulação de um LSR operando no modo de distribuição de label *DownStream On Demand*, subseção 4.2.4. O exemplo de simulação apresentado no diagrama MSC da figura 4.23, tem por objetivo observar o comportamento do LSR, diante do recebimento de uma mensagem de mapeamento de um *Peer DownStream*, para uma FEC que não foi requisitada por um *Peer UpStream*.

De acordo com a figura 4.23, o processo *StateMachine* encontra-se inicialmente no estado *Waiting* pronto para receber sinais do ambiente externo.

Envio do sinal *LDPMapping 2 D 2222 1111 900* :

O primeiro sinal enviado do ambiente ao processo *StateMachine*, é o sinal *LDPMapping* com os parâmetros (2=FEC, 2222=*peer* que enviou a mensagem (próximo *hop* para a FEC), 1111=*peer* que receberá a mensagem (LSR) e, 900=identificador da mensagem).

Ao receber um mapeamento de um *peer downstream*, o LSR verifica se há um pedido de requisição de label pendente para a FEC. Neste caso, não há um pedido pendente. Como o LSR está operando no modo de distribuição *DownStream On Demand*, ele só propaga *mappings* para FECs no qual existe uma requisição de label pendente. Isto significa que, se o *peer upstream* não requisitou um label para a FEC, o LSR não propaga *mappings*.

Portanto, não é propagado o *mapping upstream* e, o processo *StateMachine* retorna ao estado *Waiting* esperando pelo próximo sinal.

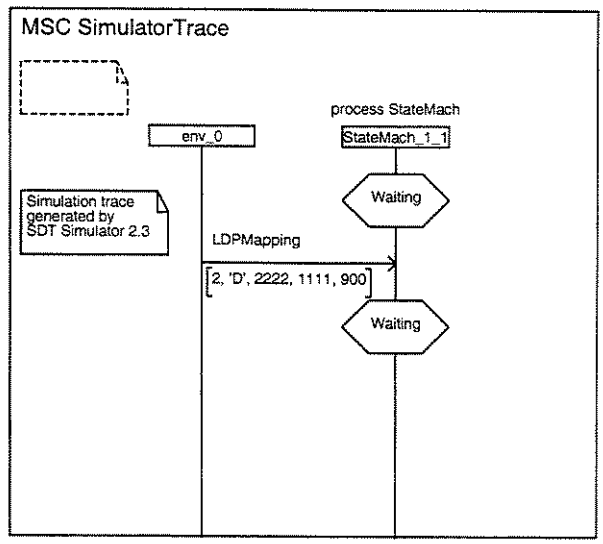

Figura 4.23: MSC Simulação de um LSR com Modo de Distribuição de Label *DownStream On Demand*

4.4.5 Simulação de um LSR operando com Retenção de Label Conservativo

Esta subseção apresenta a simulação de um LSR operando com retenção de label Conservativo. O exemplo de simulação apresentado no diagrama MSC da figura 4.24, tem por objetivo observar o comportamento do LSR, diante do recebimento de uma mensagem de mapeamento de um *Peer DownStream*, que não é próximo *hop* para nenhuma das FECs reconhecidas pelo LSR.

De acordo com a figura 4.24, o processo *StateMachine* encontra-se inicialmente no estado *Waiting* pronto para receber sinais do ambiente externo.

Envio do sinal *LDPMapping 2 D 9999 1111 900* :

O primeiro sinal enviado do ambiente ao processo *StateMachine*, é o sinal *LDPMapping* com os parâmetros (2=FEC, 9999=*peer* que enviou a mensagem (*Peer DownStream*), 1111=*peer* que receberá a mensagem (LSR) e, 900=identificador da mensagem).

Um LSR operando com modo de retenção de label conservativo, só armazena informações de *mappings* enviados de *peers downstream* que, são próximo *hop* para as FECs reconhecidas pelo LSR. Neste caso de simulação, o *peer downstream* de identificador 9999, não é próximo *hop* para nenhuma das FECs reconhecidas pelo LSR.

Desta forma, o LSR envia uma mensagem de *release* para o *peer* que enviou a mensagem de mapeamento. O envio da mensagem de *release* é representado na figura 4.24, pelo envio do sinal *LDPReleaseDown* com os parâmetros (2=FEC, D=label *downstream*, 1111=*peer* que enviou a mensagem (LSR) e, 9999=*peer* que recebeu a mensagem (*Peer DownStream*)). O processo *StateMachine* retorna ao estado *Waiting* esperando pelo próximo sinal.

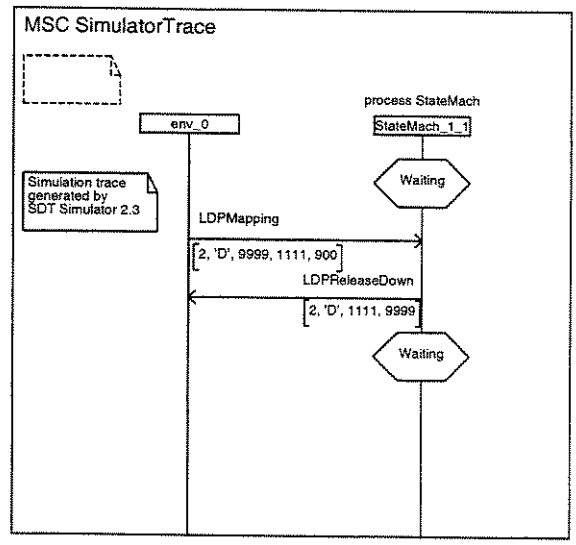

Figura 4.24: MSC Simulação de um LSR operando com Retenção de Label Conservativo

Se o LSR estivesse configurado para operar no modo de retenção de label *Liberal*, ele armazenaria as informações recebidas no mapeamento e não enviaria a mensagem de *release*, mesmo que o *peer downstream* que enviou o mapeamento, não seja um próximo *hop* para as FECs reconhecidas pelo LSR.

Com o procedimento de retenção de label *Liberal*, se houver uma mudança de próximo *hop* para uma das FECs reconhecidas pelo LSR, e ele já tiver recebido e retido um mapeamento deste próximo *hop* novo, não é preciso enviar uma requisição de label para o próximo *hop* novo.

4.5 Considerações sobre o Capítulo 4

O trabalho realizado no capítulo 4 foi de, partindo-se da máquina de estado do LDP [24] descrita na seção 4.2 e, da descrição dos procedimentos de distribuição de label do LDP [21], criar em uma única especificação formal em SDL (*Specification and Description Language*), a combinação das informações textuais apresentadas nos dois documentos.

Isto significa, especificar e simular o comportamento de um LSR que troca mensagens do LDP com seus *peers* adjacentes. Ao mesmo tempo, conforme as mensagens do LDP são enviadas e, como consequência, os LSPs vão sendo estabelecidos, são geradas as máquinas de estado do LDP, uma máquina de estado para cada LSP criado.

Através dos diagramas MSCs e, de consultas às variáveis do sistema especificado através do *Simulator* do SDT, foi possível acompanhar as ações do LSR, que ocorreram de acordo com os procedimentos de [21] como era esperado. Além disso, foram verificadas as máquinas do LDP e, as transições de estado para os LSPs criados durantes as simulações.

Entretanto, ao tentar unir as informações dos dois documentos para construir-se uma única especificação, foram encontrados pontos de discordância. Um exemplo, seria o caso

de detecção de um próximo *hop* novo para uma FEC.

De acordo com a máquina de estado do LDP [24], ao detectar-se um próximo *hop* novo para uma FEC, o LSR envia uma mensagem de requisição de label para o novo *Next-Ho* e, só envia um *release* para o antigo próximo *hop*, depois que receber um mapeamento do próximo *hop* novo.

De acordo com os algoritmos de distribuição de label descritos em [21], quando o LSR detecta uma mudança de próximo *hop* para a FEC, se ele estiver operando no modo *Conservativo*, ele já envia uma mensagem de *release* para o antigo próximo *hop*, mesmo que ainda não tenha recebido um mapeamento de label do próximo *hop* novo.

Em casos de discordância, foi adotado o procedimento descrito em [21].

Capítulo 5

Conclusão

Este trabalho propôs a especificação formal em SDL (*Specification and Description Language*) e a simulação do classificador de fluxo do IpSwitching, assim como do LDP (*Label Distribution Protocol*), o protocolo de distribuição de *label* do MPLS (*MultiProtocol Label Switching*). Estas propostas surgiram com o objetivo de se transportar o IP sobre o ATM, com a tentativa de se aliviar o congestionamento nos roteadores das redes tradicionais *hop-by-hop*, causado devido à popularização da rede Internet e, consequentemente o aumento do tráfego IP.

O que difere o IpSwitching do MPLS é que o IpSwitching utiliza uma transmissão de pacotes orientado ao tráfego (*traffic driven*), isto é, os *labels* são associados a fluxos selecionados para a comutação. Já o MPLS utiliza uma transmissão de pacotes orientado a topologia (*topology driven*), onde os *labels* são associados às FECs (*Forwarding Equivalence Class*) detectadas nas tabelas de roteamento de camada de rede.

No capítulo 3 foi apresentada a especificação formal em SDL do classificador de fluxo, que é parte integrante do controlador de um IpSwitch. A escolha do classificador de fluxo influencia diretamente no desempenho do IpSwitching. O classificador de fluxo apesar de sempre ter que ser implementado no controlador do IpSwitch, não é especificado em [14].

Através da especificação formal em SDL92 foi construída uma especificação de um analisador de fluxo simples, seção 3.2. Utilizando-se os conceitos de reutilização e de herança da orientação a objetos presente na linguagem SDL92, foi possível construir uma evolução da primeira versão, seção 3.3. A evolução, que herda toda a especificação da primeira versão, especifica um classificador de fluxo do tipo X/Y.

Duas outras versões de especificação formal foram construídas, as especificações das seções 3.4 e 3.5. A construção destas duas novas versões tinham como objetivo obter uma otimização das especificações anteriores em relação ao acesso de dados. Ao invés do acesso sequencial às tabelas, foi utilizada uma estrutura de tabela *hash*. Com a utilização das especificações otimizadas, obteve-se uma redução de mais de 50% nos tempos de simulação.

Nas simulações do classificador de fluxo através do SDT (*SDL Design Tool*) foram utilizados como alimentadores do classificador, *traces* de tráfego IP disponíveis gratuitamente por ftp em *ita.ee.lbl.gov/traces*. Através da variação dos parâmetros X, Y e do

parâmetro de fluxo inativo, foi possível coletar dados estatísticos sobre a performance do IpSwitching. Além disso, através da geração de MSCs (*Message Sequence Charts*) pôde-se avaliar o comportamento funcional do classificador diante do recebimento de pacotes, da detecção de um fluxo e, da detecção de fluxos inativos.

Conclui-se que, a configuração dos parâmetros X, Y e do parâmetro de fluxo inativo, influencia diretamente no desempenho do IpSwitching, isto é, no número de pacotes comutados no hardware de camada 2. A escolha do classificador de fluxo deve ser adequada ao tipo de tráfego e às limitações do hardware, como espaço limitado na tabela de *cross-connect*.

Em relação ao MPLS, este trabalho propôs no capítulo 4, uma expansão do diagrama da máquina de estado do LDP (*Label Distribution Protocol*) para LSRs sem capacidade de realizar *merge* sugerido em [24]. Esta expansão em 4 diagramas, além de cobrir todas as configurações de distribuição de *label*, modo de controle LSP e modo de retenção de *label*, possibilita uma maior compreensão do comportamento de um LSR (*Label Switching Router*) de acordo com a sua configuração, diante do estabelecimento de LSPs (*Label Switched Paths*). Isto significa que, para cada configuração o LSR toma diferentes ações que podem ser visualizadas separadamente em cada um dos diagramas.

As informações textuais sobre a máquina de estado e sobre os procedimentos de distribuição de *label* do LDP foram combinadas na seção 4.3, em uma única especificação formal em SDL.

Através de simulações da especificação do capítulo 4, foi possível gerar MSCs que analisavam o comportamento de um LSR diante da troca de mensagens LDP com seus *peers* adjacentes e, analisar a situação das máquinas de estado LDP para cada um dos LSPs a serem estabelecidos.

Em geral, tanto no capítulo 3 como no capítulo 4, a utilização da linguagem SDL trouxe como contribuição, a possibilidade de construir-se uma especificação formal do classificador de fluxo, assim como a especificação formal do LDP partindo-se de informações textuais.

As características de orientação a objetos, possibilitaram a criação de uma evolução de especificação. As simulações utilizando o SDT foram realizadas com duas abordagens. A primeira abordagem, a especificação do classificador de fluxo do capítulo 3, teve como objetivo analisar o desempenho do protocolo IpSwitching, considerando-se a porcentagem de pacotes comutados em relação ao número total de pacotes processados pelo controlador. A segunda abordagem, a especificação dos procedimentos e da máquina de estado do LDP, teve como objetivo analisar o comportamento funcional do protocolo, e não o seu desempenho.

5.1 Trabalhos Futuros

Neste trabalho foi analisado o desempenho do IpSwitching em relação ao comportamento funcional do classificador de fluxo e, em relação a configuração dos parâmetros do

classificador. Através de simulações da especificação formal e da utilização de *traces* de tráfego IP, foi possível coletar dados estatísticos sobre o desempenho do IpSwitching.

Neste trabalho, quando foi feita referência ao desempenho do IpSwitching, tratava-se da análise da porcentagem de pacotes comutados em relação do número total de pacotes processados pelo controlador do IpSwitch, da mesma forma em que as simulações do IpSwitching são abordadas em [23]. Entretanto, o trabalho apresentado no capítulo 3 preocupa-se não só com a análise do desempenho do IpSwitching, mas com o comportamento funcional do classificador de fluxo.

Uma sugestão para trabalhos futuros sobre o IpSwitching seria, através de uma ferramenta apropriada de análise de performance de protocolos de redes, analisar a performance do IpSwitching levando em consideração, não só o classificador de fluxo, mas também as mensagens enviadas pelo IFMP (*Ipsilon Flow Management Protocol*). De acordo com a configuração do classificador, seria possível analisar o número de mensagens do IFMP necessárias para o estabelecimento, manutenção e remoção de SVCs (*Switched Virtual Connection*).

Em relação ao MPLS, uma sugestão para trabalhos futuros que poderiam reutilizar o código de especificação do LSR apresentado no capítulo 4, seria a especificação de uma rede de LSRs que trocam mensagens LDP e pacotes de camada de rede entre si. Esta evolução de especificação poderia ser construída em SDL da seguinte maneira :

- através da utilização de várias instâncias do bloco (*LabelSwitchRouter*) da figura 4.10 do capítulo 4, seria possível criar uma rede de nós MPLS.

- as adjacências de cada nó MPLS estariam configuradas através de canais no SDL, que representariam as sessões TCP de um LSR com seus LDP *peers downstream* e *upstream*.

Estes nós MPLS seriam capazes de trocar mensagens LDP entre si e além disso, estabelecidos LSPs, poderia ser simulado o envio de pacotes com *label* e sem *label*. Cada LSR poderia ser configurado para operar com diversas configurações de distribuição de *label*.

O objetivo de se criar uma especificação de uma rede de nós MPLS é a possibilidade de simular situações de *loop* na rede. Hoje, o método de prevenção de *loop* foi retirado da versão 2 da especificação do LDP [21].

O primeiro documento de especificação do LDP (versão 1) apresentava a proposta do método de prevenção de *loop* através da difusão. Entretanto, outro método de prevenção de *loop* baseado em *threads* foi proposto em [35].

Através da especificação dos dois mecanismos, seria possível nas simulações, alterar as tabelas de roteamento de cada LSR, criando situações de *loop* que fossem testados pelos dois mecanismos de prevenção. Uma análise de desempenho poderia ser obtida comparando-se o número de mensagens necessárias para realizar a prevenção de *loop*, tanto pelo mecanismo de difusão como pelo mecanismo que utiliza *threads*.

Bibliografia

[1] ATM User-Network Interface (UNI) Specification, Version 3.0. af-uni-0010.002, ATM Forum, Setembro 1993.

[2] ATM User-Network Interface (UNI) Specification, Version 3.1. af-uni-0010.002, ATM Forum, Setembro 1994.

[3] LAN Emulation Over ATM Version 1.0. af-lane-0021.000, ATM Forum, Jan 1995.

[4] ATM User-Network Interface (UNI) Signaling Specification, Version 4.0. af-sig-0061.000, ATM Forum, Julho 1996.

[5] PNNI Specification Version 1.0. af-pnni-055.000, ATM Forum, Março 1996.

[6] LAN Emulation Over ATM Version 2.0 - LUNI Specification. af-lane-0084.000, ATM Fórum, Julho 1997.

[7] MultiProtocol Over ATM (MPOA) versão 1.0. af-mpoa-0087.000, ATM Forum, Julho 1997.

[8] A. Alles. ATM Internetworking. White paper, http://www.cisco.com/warp/public/614/12.html, Cisco Systems, Inc, Maio 1995.

[9] G. Armitage. Support for Multicast over UNI 3.0/3.1 based ATM Networks. Rfc 2022, IETF, Novembro 1996.

[10] Zheng Wang Arun Viswanathan, Nancy Feldman and Ross Callon. Evolution of Multiprotocol Label Switching. *IEEE Communications Magazine*, 36(5):165–173, Maio 1998.

[11] K.McCloghrie Y.Rekhter E.Rosen. G.Swallow e P.Doolan B.Davie, J.Lawrence. Use of Label Switching With ATM. internet-draft, IETF-Network Working Group, Setembro 1998.

[12] Philip Dumortier. Toward a New IP over ATM Routing Paradigm. *IEEE Communications Magazine*, 36(1):82–86, Jan 1998.

[13] N.Feldman e A.Viswanathan. ARIS Specification. *draft-feldman-aris-spec-00.txt*, IETF, Setembro 1997.

[14] P.Newman W. L. Edwards R. Hinden E. Hoffman F. Ching Liaw T. Lyon e G. Minshall. Ipsilon Flow Management Protocol Specification for IPv4, Versão 1.0. Rfc1953, IETF-Network Working Group, Maio 1996.

[15] Arun Viswanathan e Ross Callon Eric C. Rosen. Multiprotocol Label Switching Architecture. draft-ietf-mpls-arch-02.txt, IETF - MPLS Working Group, Julho 1998.

[16] ITU-T(CCITT Recommendation Z.100), Geneve, Switzerland. *Specification and Description Language*, 1993.

[17] ITU-T(CCITT Recommendation Z.120), Geneve, Switzerland. *Message Sequence Chart(MSC)*, 1993.

[18] D.Piscitello e B.Cole J. Luciani, D.Katz. NBMA Next Hop Resolution Protocol (NHRP). draft-ietf-rolc-nhrp-11.txt, IETF, Março 1997.

[19] Y.Shobatake A.Mogi S.Matsuzawa T.Jinmei e H.Esaki K.Nagami, Y.Katsube. Toshiba's Flow Attribute Notification Protocol (FANP) Specification. Rfc2129, IETF, Abril 1997.

[20] G. Lemos e S. Colcher L. F. G. Soares. *Redes de Computadores: Das LANs, MANs e WANs às Redes ATM.* Editora Campus, Rio de Janeiro, 1995.

[21] N.Feldman A.Fredette L.Anderson, P.Doolan and B.Thomas. LDP Specification. draft-ietf-mpls-ldp-02.txt, IETF - MPLS Working Group, Nov 1998.

[22] M. Laubach. Classical Ip and ARP over ATM. Rfc1577, IETF, Jan 1994.

[23] Steven Lin and Nick McKeown. A Simulation Study of IP Switching. *Technical Report CSL-TR-97-720, Ipsilon Corporation*, Abril 1997.

[24] Pierrick Cheval Liwen Wu and Christophe Boscher. LDP State Machine. draft-wu-mpls-ldp-state-00.txt, IETF - MPLS Working Group, Outubro 1998.

[25] Eric Mannie. Multiprotocol Over ATM : Models and Comparison. In *2nd ATM Symposium, ATM, IP and High Speed Networks*, Brussels-Belgian, November,1997.

[26] T. Lyon P. Newman and G. Minshall. Flow labelled IP : A connectionless approach to ATM. In *Proceedings of IEEE Infocom*, Março,1996.

[27] Greg Minshall Peter Newman and Larry Huston. Ip Switching : ATM Under IP. *IEEE/ACM Transactions on Networking*, 6(2):117–129, Abril 1998.

[28] R.Hinden E.Hoffman F.Liaw T. Lyon e G.Minshall P.Newman, W.Edwards. Ipsilon General Switch Management Protocol Specification, Versão 2.0. Rfc2297, IETF, Março 1998.

[29] T.Lyon e G. Minshall P.Newman, W.L.Edwards R.Hinden E.Hoffman F.C.Liaw. Transmission of Flow Labelled IPv4 on ATM Data Links. Rfc1954, IETF-Network Working Group, Maio 1996.

[30] N.Feldman A.Fredette G.Swallow e A. Viswanathan R.Callon, P.Doolan. A Framework for Multiprotocol Label Switching. draft-ietf-mpls-framework-02.txt, IETF - MPLS Working Group, Nov 1997.

[31] George C. Sackett and Christopher Y. Metz. *ATM and Multiprotocol Networking.* McGraw-Hill Companies, New York, NY, 1996.

[32] Telelogic AB, Malmö, Sweden. *Telelogic ORCA and SDT 3.3, Getting Started, Part 1: Basic Tools Tutorials*, Fevereiro,1998.

[33] Telelogic AB, Malmö, Sweden. *Telelogic ORCA and SDT 3.3, Methodology Guidelines, Part2 : Practical SDL Guidelines*, Fevereiro,1998.

[34] K.Nagami e H.Esaki Y.Katsube. Toshiba's Router Architecture Extensions for ATM : Overview. Rfc2098, IETF, Fevereiro 1997.

[35] Eric Rosen e Paul Doolan Yoshihiro Ohba. MPLS Loop Prevention Mechanism. draft-ohba-mpls-loop-prevention-02.txt, MPLS Working Group, Novembro 1998.

[36] D.Katz E.Rosen e G.Swallow Y.Rekhter, B.Davie. Cisco Systems' Tag Switching Architecture Overview. Rfc2105, IETF, Fevereiro 1997.

Apêndice A

Descrição da Saída do Simulador do SDT para o Sistema MPLS

Este apêndice apresenta uma descrição da saída do *Simulator* do SDT para o sistema MPLS especificado na seção 4.3, isto é, o formato no qual é apresentado os valores atribuídos a variáveis do sistema MPLS.

Os valores das variáveis podem ser obtidos através de consultas, no *Simulator* do SDT, durante as simulações. Os valores das variáveis são apresentados da seguinte forma : nome da variável, seguido pelo tipo da variável entre parênteses, o sinal de igual e, finalmente o valor da variável.

No caso da simulação do sistema MPLS, as variáveis *Routing*, *LSPCB*, *UpLabel* e *Message* são dos tipos *RoutingTable*, *LSPCBTable*, *UpLabelTable* e *MessageTable*. Estes quatro tipos são definidos como vetores de estruturas, como descrito na figura 4.9.

A seguir apresenta-se os resultados das variáveis após a execução do procedimento de inicialização do sistema MPLS, onde as variáveis de configuração, a tabela de roteamento e, a tabela de labels são inicializados. Estes resultados não são considerados como resultados de simulação e, são mostrados neste apêndice apenas como um exemplo. Foi rodado o procedimento *Initialization*, porque ele é o procedimento que entra em ação quando o sistema MPLS é inicializado, figura 4.13. Estes resultados são apenas ilustrativos para que todas as variáveis do sistema MPLS possam ser apresentadas.

O texto em itálico apresenta a saída do simulador. O texto em *type writer* apresenta os possíveis valores assumidos pela variável, quando forem limitados ou, uma breve descrição da função da variável.

LabDisProc (natural) = 1
1 - `DownStream Unsolicited Independent`
2 - `DownStream Unsolicited Ordered`
3 - `DownStream On Demand Independent`
4 - `DownStream On Demand Ordered`

LSRIdentifier (natural) = 1111

```
Identificador Ip do LSR Simulado

LabelRetention (natural) = 1
1- Conservative, 2- Liberal

UpLDPPeer (natural) = 123
Identificador do LDP Peer UpStream

Routing (RoutingTable) = (1:(. 1, 0, 1 .)), (2:(. 2, 2222, 2 .))
 Campos da tabela de roteamento
 (FEC), (Ip do Next-Hop), (Egress)
 Egress = 1, No Egress = 2

UpLDPPeer (natural) = 123
Identificador do LDP Peer UpStream

LSPCB (LSPCBTable) = (: (others:(. 0, 0, 0, 0, 0, 0, '', '', 0, 0 .)) :)
  Campos da tabela de Controle LSP
  (ID da Mensagem Request enviada por um LDP Peer UpStream)
  (ID da Mensagem Request enviada pelo LSR ao Next-Hop para a FEC)
  (Identificador Ip do LDP Peer UpStream)
  (Identificador Ip do LDP Peer DownStream (Next-Hop for FEC))
  (Estado do LSP)
    1 - Idle
    2 - Response Awaited
    3 - Established
    4 - New NH Response Awaited
  (FEC)
  (UpStream Label)
  (DownStream Label)
  (Identificador do novo Next-Hop para a FEC)
  (Identificador da mensagem de Request enviada para o novo Next-Hop)

UpLabel (UpLabelTable) = (1:(. 'A', 'A' .)), (2:(. 'B', 'A' .)), (3:(. 'C', 'A' .))
Campos da tabela de labels UpStream (Label, Status do Label)
'A' - Available, 'B' - USED

Message (MessageTable) = (: (others:(. '', 0, 0, 0, '', 0, '' .)) :)
  Campos da Tabela de Registro de Mensagens LDP

  (Tipo da Mensagem)
    'R' - Request
    'M' - Mapping
```

'W' - WithDraw
'Re'- Release
'NR' - No Route (Notification)
'NLR' - No Lable Resource (Notification)
(Identificador do peer que enviou a mensagem)
(Identificador do peer que recebeu a mensagem)
(FEC)
(Lable)
(ID da mensagem transmitida)
(Status da mensagem)
'P' - Pending
'E' - Established
'S' - Sent
'I' - Independent

GenerateID (natural) = 800

Variável que gera identificadores únicos para as mensagens enviadas pelo LSR aos seus peers.

Apêndice B

Resultados detalhados da Simulação de um LSR operando com Modo de Controle Independente

Este apêndice apresenta uma descrição detalhada dos resultados obtidos na simulação de um LSR configurado para operar no modo de controle LSP Independente. A simulação é apresentada através do diagrama MSC (representando os sinais enviados do ambiente externo ao sistema e, do sistema ao ambiente externo), da figura B.1.

Além disso, os eventos ocorridos (envio de sinais) e, o resultados das variáveis do sistema após a ocorrência do evento, são apresentadas nas tabelas B.1 e B.2. Os eventos são numerados de 1 a 10, sendo os eventos de 1 a 5 apresentados na tabela B.1 e, a continuação dos eventos de 6 a 10, na tabela B.2.

As tabelas B.1 e B.2, apresentam na primeira coluna, os eventos na ordem em que ocorreram durante a simulação, eventos da figura B.1 e, na segunda coluna, a situação das variáveis do sistema após a ocorrência dos eventos da primeira coluna.

Primeiro Evento (Inicialização) : o primeiro evento ocorrido, tabela B.1, foi o de inicialização do sistema MPLS.

Após a ocorrência do evento 1, o estado das variáveis apresentado na segunda coluna da tabela B.1 é :

- variável *LabDisProc* foi inicializada com o valor 1, isto é, o LSR está configurado para operar no modo *DownStream* Independente.

- o identificador do LSR, *LSRIdentifier*, foi inicializado com o valor simbólico 1111, representando seu endereço Ip.

- a variável *LabelRetention* foi inicializada com o valor 2 indicando que o LSR operará no modo de retenção de label Liberal.

- a variável *UpLDPPeer* é inicializada com o valor simbólico 123 representando o endereço Ip do LDP *Peer UpStream.*

- a tabela de roteamento, *Routing*, foi inicializada com dois registros. O primeiro registro armazena as informações (1=FEC , 0= *Next-Hop*, 1=o LSR é *egress* para a FEC 1). O segundo registro armazena as informações, (2=FEC, 2222=identificador Ip do *Next-Hop* para a FEC, 2=o LSR é *egress* para a FEC 2).

- a tabela de labels, *UpLabel*, é inicializada com 5 registros. Cada registro apresenta o label e o seu status. Desta forma os labels *A, B, C, D* e *E*, estão todos disponíveis quando o sistema é inicializado, status = A (*Available*).

Após a inicialização, o sistema vai para o estado *Waiting*, figura B.1 e, está pronto para a ocorrência do próximo evento, ou seja, do envio do próximo sinal. Sempre que o sistema estiver no estado *Waiting*, ele está pronto para o recebimento de sinais.

Segundo Evento (*LDPRequest 1 123 1111 11*) : o segundo evento ocorrido, tabela B.1, é representado pelo envio do sinal *LDPRequest* ao sistema MPLS com os parâmetros (1=FEC, 123=identificador Ip do LDP *Peer UpStream* que está enviando o *request*, 1111=identificador Ip do *peer* que receberá o *request*, 11=identificador da mensagem). O envio do sinal pode ser observado no diagrama MSC da figura B.1 na sequência de ocorrência de eventos.

Ao consultar a tabela de roteamento, o LSR verifica que ele é *egress* para a FEC 1. De acordo com a máquina de estado do LDP, se o LSR for *egress* para a FEC, ele aloca um label *upstream* disponível e envia uma mensagem de *mapping* para o LDP *Peer Upstream* que requisitou o label.

Pode-se observar na tabela B.1, na coluna estado das variáveis após a ocorrência do segundo evento que :

- o primeiro registro da tabela de labels, *UpLabel*, teve o segundo campo alterado de A (*Available*) para U (*Used*),

- na tabela *Message*, foi registrado o envio da mensagem de *mapping* no primeiro registro da tabela (M=*mapping*, 1111=*peer* que enviou a mensagem, 123=*peer* que receberá a mensagem, 11=identificador do *request* previamente enviado, E=*Established*, 1=FEC e A=label *upstream* alocado).

- foram armazenadas as informações do estabelecimento do LSP para a FEC 1, através da criação de um registro na tabela de *LSPCB*. As informações foram armazenadas no primeiro registro da tabela (11=identificador do *request* enviado pelo LDP *Peer UpStream*, 0=identificador da mensagem *DownStream*, 123=identificador do LDP *Peer UpStream*, 0=*Next-Hop*, 3=*Established*, 1=FEC, A=label *upstream*, EGRESS=label *downstream*, 0=novo *Next-Hop*, 0=identificador de *request* para o novo *Next-Hop*). O LSP para a FEC 1 passou do estado *Idle* para o estado *Established*. A associação do label *downstream* com o label *upstream*, é representado utilizando-se a palavra *EGRESS* como label *downstream*, para indicar que o LSR é *egress* para a FEC.

Evento	Estado das Variáveis
1 - Inicialização	LabDIsProc (natural) = 1 LSRIdentifier (natural) = 1111 LabelRetention (natural) = 2 UpLDPPeer (natural) = 123 Routing (RoutingTable) = (1:(. 1, 0, 1 .)), (2:(. 2, 2222, 2 .)) UpLabel (UpLabelTable) = (1:(. 'A', 'A' .)), (2:(. 'B', 'A' .)), (3:(. 'C', 'A' .)), (4:(. 'D', 'A' .)), (5:(. 'E', 'A' .))
2 - *LDPRequest* 1 123 1111 11	LSPCB (LSPCBTable) = (1:(. 11, 0, 123, 0, 3, 1, 'A', 'EGRESS', 0, 0 .)) UpLabel (UpLabelTable) = (1:(. 'A', 'U' .)), (2:(. 'B', 'A' .)), (3:(. 'C', 'A' .)), (4:(. 'D', 'A' .)), (5:(. 'E', 'A' .)) Message (MessageTable) = (1:(. 'M', 1111, 123, 11, 'E', 1, 'A' .))
3 - *LDPRequest* 2 123 1111 12 – :	LSPCB (LSPCBTable) = (1:(. 11, 0, 123, 0, 3, 1, 'A', 'EGRESS', 0, 0 .)), (2:(. 12, 800, 123, 2222, 2, 2, 'B', '', 0, 0 .)) UpLabel (UpLabelTable) = (1:(. 'A', 'U' .)), (2:(. 'B', 'U' .)), (3:(. 'C', 'A' .)), (4:(. 'D', 'A' .)), (5:(. 'E', 'A' .)) Message (MessageTable) = (1:(. 'M', 1111, 123, 11, 'E', 1, 'A' .)), (2:(. 'M', 1111, 123, 12, 'E', 2, 'B' .)) (3:(. 'R', 1111, 2222, 800, 'I', 2, '' .))
4 - *LDPMapping* 2 'X' 2222 1111 800	LSPCB (LSPCBTable) = (1:(. 11, 0, 123, 0, 3, 1, 'A', 'EGRESS', 0, 0 .)), (2:(. 12, 800, 123, 2222, 2, 2, 'B', 'X', 0, 0 .)) Message (MessageTable) = (1:(. 'M', 1111, 123, 11, 'E', 1, 'A' .)), (2:(. 'M', 1111, 123, 12, 'E', 2, 'B' .)) (3:(. 'M', 2222, 1111, 800, 'E', 2, 'X' .))
5 - IntNewNH 2 4444 2222 'X'	Routing (RoutingTable) = (1:(. 1, 0, 1 .)), (2:(. 2, 4444, 2 .)) LSPCB (LSPCBTable) = (1:(. 11, 0, 123, 0, 3, 1, 'A', 'EGRESS', 0, 0 .)), (2:(. 12, 800, 123, 2222, 4, 2, 'B', '', 4444, 801 .)) Message (MessageTable) = (1:(. 'M', 1111, 123, 11, 'E', 1, 'A' .)), (2:(. 'M', 1111, 123, 12, 'E', 2, 'B' .)), (3:(. 'Re', 1111, 2222, 801, 'S', 2, 'X' .)), (4:(. 'R', 1111, 4444, 801, 'I', 2, '' .))

Tabela B.1: Eventos e Situação das Variáveis, MSC da figura B.1 (LSR Modo de Controle LSP Independente)

Evento	Estado das Variáveis
6 - *LDPMapping* 2 'Y' 4444 1111 801	LSPCB (LSPCBTable) = (1:(. 11, 0, 123, 0, 3, 1, 'A', 'EGRESS', 0, 0 .)), (2:(. 12, 801, 123, 4444, 3, 2, 'B', 'Y', 0, 0 .)) Message (MessageTable) = (1:(. 'M', 1111, 123, 11, 'E', 1, 'A' .)), (2:(. 'M', 1111, 123, 12, 'E', 2, 'B' .)) (3:(. 'Re', 1111, 2222, 801, 'S', 2, 'X' .)), (4:(. 'M', 4444, 1111, 801, 'E', 2, 'Y' .))
7 - LDPWithDraw 2 'Y' 4444 1111	LSPCB (LSPCBTable) = (1:(. 11, 0, 123, 0, 3, 1, 'A', 'EGRESS', 0, 0 .)), (2:(. 12, 801, 123, 4444, 3, 2, 'B', '', 0, 0 .)) Message (MessageTable) = (1:(. 'M', 1111, 123, 11, 'E', 1, 'A' .)), (2:(. 'M', 1111, 123, 12, 'E', 2, 'B' .)), (3:(. 'Re', 1111, 2222, 801, 'S', 2, 'X' .)), (4:(. 'Re', 1111, 4444, 0, 'S', 2, 'Y' .)), (5:(. 'W', 1111, 123, 0, 'P', 2, 'B' .))
8 - *LDPRelease* 2 'B' 123 1111	LSPCB (LSPCBTable) = (1:(. 11, 0, 123, 0, 3, 1, 'A', 'EGRESS', 0, 0 .)) UpLabel (UpLabelTable) = (1:(. 'A', 'U' .)), (2:(. 'B', 'A' .)), (3:(. 'C', 'A' .)), (4:(. 'D', 'A' .)), (5:(. 'E', 'A' .)) Message (MessageTable) = (1:(. 'M', 1111, 123, 11, 'E', 1, 'A' .)), (2:(. 'Re', 1111, 2222, 801, 'S', 2, 'X' .)), (3:(. 'Re', 1111, 4444, 0, 'S', 2, 'Y' .))
9 - *LDPMapping* 2 'Z' 4444 1111 90	LSPCB (LSPCBTable) = (1:(. 11, 0, 123, 0, 3, 1, 'A', 'EGRESS', 0, 0 .)) (2:(. 802, 90, 123, 4444, 3, 2, 'B', 'Z', 0, 0 .)) UpLabel (UpLabelTable) = (1:(. 'A', 'U' .)), (2:(. 'B', 'U' .)), (3:(. 'C', 'A' .)), (4:(. 'D', 'A' .)), (5:(. 'E', 'A' .)) Message (MessageTable) = (1:(. 'M', 1111, 123, 11, 'E', 1, 'A' .)), (2:(. 'Re', 1111, 2222, 801, 'S', 2, 'X' .)), (3:(. 'Re', 1111, 4444, 0, 'S', 2, 'Y' .)) (4:(. 'M', 4444, 1111, 90, 'E', 2, 'Z' .)), (5:(. 'M', 1111, 123, 0, 'P', 2, 'B' .))
10 - *LDPMapping* 2 'G' 5555 1111 91	Message (MessageTable) = (1:(. 'M', 1111, 123, 11, 'E', 1, 'A' .)), (2:(. 'Re', 1111, 2222, 801, 'S', 2, 'X' .)), (3:(. 'Re', 1111, 4444, 0, 'S', 2, 'Y' .)) (4:(. 'M', 4444, 1111, 90, 'E', 2, 'Z' .)), (5:(. 'M', 1111, 123, 0, 'P', 2, 'B' .)) (6:(. 'M', 5555, 1111, 91, 'P', 2, 'G' .))

Tabela B.2: Eventos e Situação das Variáveis Continuação, MSC da figura B.1 (LSR Modo de Controle LSP Independente)

O envio da mensagem de *mapping* pode ser observado na sequência, figura B.1, representado pelo envio do sinal *LDPMapping* com os parâmetros (1=FEC, A=label alocado, 1111=identificador do *peer* está enviando a mensagem, 123=identificador do *peer* que receberá a mensagem, 11=identificador do *request* previamente enviado e, *NOT PROPAGATING* porque o LSR não está propagando uma mensagem de *mapping* recebida de um *peer downstream*).

Terceiro Evento (*LDPRequest 2 123 1111 12*) : o terceiro evento ocorrido, tabela B.1, é representado pelo envio do sinal *LDPRequest*, sequência da figura B.1, com os parâmetros (2=FEC, 123=identificador do LDP *Peer UpStream* requisitante, 1111= identificador do LSR e, 12=identificador da mensagem).

O LSR está configurado como não *egress* para a FEC=2. Desta forma, de acordo com a máquina de estado do LDP, o LSR deve verificar se já recebeu um label de um LDP *Peer DownStream* para a FEC. Neste caso, não recebeu. Como o LSR está operando no modo independente, mesmo que não tenha recebido um label *downstream*, ele deve alocar um label *upstream* e enviar uma mensagem de *mapping* para o LDP *Peer UpStream* requisitante. Além disso, ele deve propagar o pedido de *request* para o *Next-Hop* para a FEC, neste caso o LDP *Peer DownStream* de identificador 2222.

Pode-se observar na tabela B.1, na coluna de estado das variáveis, a situação das variáveis após a ocorrência do evento número 3 :

- foi criado um novo registro de LSP na tabela *LSPCB*, registro de número 2 (12=identificador do *request upstream*, 800=identificador do *request downstream*, 123=identificador do LDP *Peer UpStream*, 2222=identificador do *Next-Hop* para a FEC, 2=FEC, 2=*Response Awaited*, B=label *upstream*, " label *downstream*, 0=novo *Next-Hop*, 0=identificador de *request* para o novo *Next-Hop*). O LSP para a FEC 2 passou do estado *Idle* para o estado *Response Awaited* e permanecerá neste estado, até que receba um *mapping* do *Next-Hop* para a FEC.

- na tabela *UpLabel*, o status do segundo registro, referente ao label B, foi alterado de A (*Available*) para U (*Used*), indicando que o label B foi alocado.

- na tabela de mensagens, *Message* dois novos registros foram inseridos. O registro de número 2 armazena : (M=*mapping*, 1111=identificador do *peer* enviou a mensagem (LSR), 123=identificador do *peer* que receberá a mensagem (*Peer UpStream*), 12=identificador do *request* previamente recebido pelo LSR, E=*Established*, 2=FEC e B=label *upstream* alocado). O registro de número 3 passa a armazenar : (R=*request*, 1111=*peer* que enviou a mensagem (LSR), 2222=*peer* que receberá a mensagem (*Peer DownStream*), 800=identificador da mensagem, I=pendente e independente, 2=FEC, "=label).

Pode-se observar na sequência de sinais da figura B.1 o envio do sinal *LDPMapping* ao ambiente externo ao processo *StateMachine* com os parâmetros (2=FEC, B=label

upstream alocado, 1111=*peer* que enviou a mensagem, 123=*peer* requisitante, 12=identificador da mensagem de *request* previamente enviada, *NOT PROPAGATING* porque o LSR não está propagando um *mapping* enviado por um *peer downstream*).

Quarto Evento (*LDPMapping 2 X 2222 1111 800*)O quarto evento, tabela B.1, é representado pelo envio do sinal *LDPMapping* com os parâmetros (2=FEC, X=label *downstream*, 2222=identificador do *peer* que enviou a mensagem (*Next-Hop* para a FEC 2), 1111=identificador do *peer* que receberá a mensagem (LSR), 800=identificador da mensagem de *request* previamente enviada por 1111 a 2222).

Estando o LSP para a FEC 2 no estado *Response Awaited*, ao receber a mensagem de *mapping* do *peer Next-Hop* para a FEC, em resposta ao *request* pendente previamente enviado, o LSR atualiza o LSPCB para a FEC 2 e, armazena as informações da mensagem de *mapping* recebida.

A situação das variáveis após a ocorrência do evento de número 4, pode ser observada na quarta linha e segunda coluna da tabela B.1 :

- o segundo registro da tabela *LSPCB*) foi modificado, alterando-se o quinto campo, de 2 (*Response Awaited*), para 3 (*Established*) e, inserindo-se no oitavo campo, o label recebido X.

- a mensagem de *request* que estava armazenada no registro 3 da tabela *Message*, é removida da tabela. A mensagem de *mapping* recebida é inserida na tabela de mensagens no terceiro registro (M=*mapping*, 2222=identificador do *peer* que enviou a mensagem (*Next-Hop* para a FEC 2), 1111=identificador do *peer* que recebeu a mensagem (LSR), 800=identificador da mensagem de *request* previamente enviada, E=*Established*, 2=FEC, X=label *downstream*).

Como o LSR está operando no modo independente, ele já tinha enviado um label *upstream* para o LDP *Peer UpStream*, ao receber a mensagem de *request* para a FEC=2. Desta forma, o *mapping* não é propagado *upstream*.

Quinto Evento (*IntNewNH 2 4444 2222 X*)O quinto evento, tabela B.1, é representado pelo envio do sinal *IntNewNH* com os parâmetros (2=FEC, 4444=Novo *Next-Hop* para a FEC, 2222=Antigo *Next-Hop* para a FEC, X=Antigo Label *downstream*). De acordo com a máquina de estado do LDP, ao ocorrer o evento de detecção de um novo *Next-Hop* representado pelo envio do sinal *IntNewNh*, o LSP passa para o estado *New NH Response Awaited* porque, apesar de estar operando no modo de retenção de label liberal, o LSR não recebeu previamente nenhum mapping para a FEC 2 do novo *Next-Hop*. A ação do LSR em resposta ao evento é de enviar uma mensagem de *release* para o antigo *Next-Hop* e, uma mensagem de *request* para o novo *Next-Hop*

A situação das variáveis após a ocorrência do evento *IntNewNH* é apresentada na tabela B.1 :

- o segundo registro da tabela *Routing*, teve o terceiro campo alterado de 2222 para 4444, que é o novo *Next-Hop* para a FEC 2.

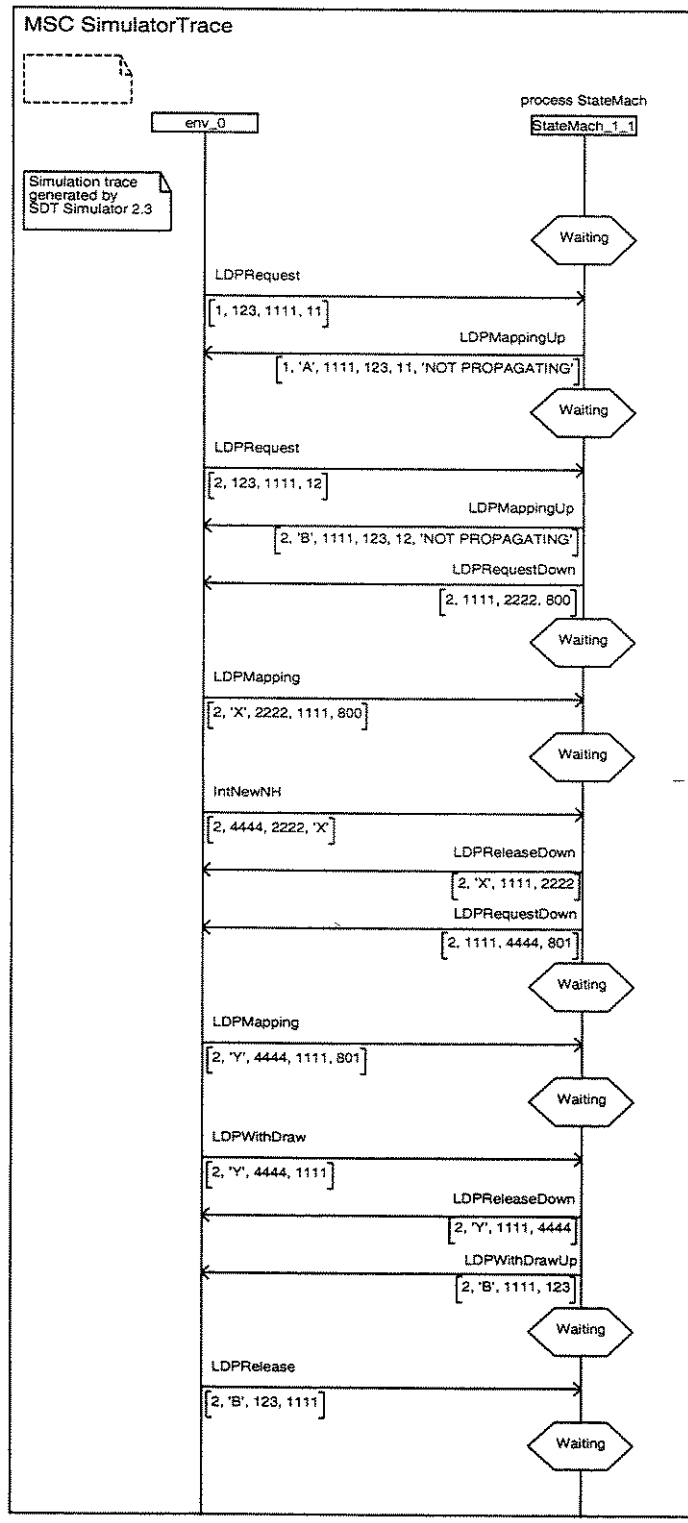

Figura B.1: MSC Simulação da Máquina de Estado do LDP, LSR Modo de Controle LSP Independente

- o segundo registro da tabela *LSPCB*, teve o quinto campo alterado de 3 (*Established*), para 4 (*New NH Response Awaited*). O oitavo campo foi reinicializado liberando o label *downstream* X. O nono campo recebeu o endereço do novo *Next-Hop* para a FEC e, o décimo campo recebeu o identificador da mensagem de *request* enviada para o novo *Next-Hop*.

- a tabela de mensagens, *Message*, teve o terceiro registro removido. O terceiro registro armazenava informações sobre o *mapping* recebido do antigo *Next-Hop* com o label *downstream* X. Como o LSR não se utiliza mais deste label, o registro da mensagem de *mapping* é removido da tabela. A mensagem de *request* enviada para o novo *Next-Hop*, é armazenada na tabela de mensagens no registro 3 (R=*request*, 1111=*peer* que enviou a mensagem (LSR), 4444=*peer* que receberá a mensagem (novo *Next-Hop* para a FEC), 801=identificador da mensagem, I=pendente/independente, 2=FEC, "=label). O LSP para a FEC 2 permanecerá no estado *New NH Response Awaited*, até que receba um *mapping* do novo *Next-Hop* para a FEC.

Observando-se a sequência da figura B.1, o LSR envia uma mensagem de *release* para o antigo *Next-Hop*, representado pelo envio do sinal *LDPRelease* do processo *StateMachine* para o ambiente externo com os parâmetros (2=FEC, X=label *downstream*, 1111=*peer* que está enviando a mensagem (LSR), 2222=identificador do antigo *Next-Hop* para a FEC). Além disso, o LSR envia uma mensagem de *request* para o novo *Next-Hop*, representado pelo envio do sinal *LDPRequestDown* do processo *StateMachine* para o ambiente externo com os parâmetros (2=FEC, 1111=*peer* que enviou a mensagem (LSR), 4444=identificador do novo *Next-Hop* para a FEC, 801=identificador da mensagem).

Sexto Evento (*LDPMapping 2 Y 4444 1111 801*)O sexto evento, tabela B.2, é representado pelo envio do sinal *LDPMapping* com os parâmetros (2=FEC, Y=label *downstream*, 4444=*peer* que enviou a mensagem (novo *Next-Hop*), 1111=*peer* que receberá a mensagem (LSR), 801=identificador da mensagem de *request* previamente enviada ao *peer* 4444 pelo *peer* 1111).

Estando o LSP para a FEC 2 no estado *New NH Response Awaited*, ao receber uma mensagem de *mapping* do novo *Next-Hop*, o LSR, de acordo com a máquina de estado do LDP, realiza o procedimento de *Local Repair* e, retorna ao estado *Established*.

A situação das variáveis após a ocorrência do evento de número 7 é mostrado na tabela B.1, na coluna de estado das variáveis :

- na tabela *LSPCB*, o segundo registro referente à FEC 2, teve o segundo campo alterado de 800 para 801 que é o novo identificador da mensagem de *request* para o novo *Next-Hop*. O quarto campo foi modificado de 2222 para o endereço do novo *Next-Hop*, 4444. O quinto campo foi alterado de 4 *New NH Response Awaited*, para 3 *Established*. O oitavo campo recebeu o novo label *downstream*, label Y, enviado na mensagem de *mapping* pelo novo *Next-Hop* e, os dois últimos campos foram reinicializados com o valor zero.

- na tabela de mensagens, *Message*, o terceiro registro referente ao *request* enviado de 1111 para 4444 foi removido, já que o pedido foi atendido com o recebimento de um *mapping* do novo *Next-Hop* para a FEC 2. A mensagem de *mapping* recebida do novo *Next-Hop* com o label downstream Y, foi armazenado na terceira posição da tabela.

Sétimo Evento (*LDPWithDraw 2 Y 4444 1111*)O sétimo evento, tabela B.2, é representado pelo envio do sinal LDPWithDraw com os parâmetros (2=FEC, Y=label *downstream* a ser removido, 4444=*peer* que enviou a mensagem (*Next-Hop* para a FEC 2), 1111=*peer* que receberá a mensagem (LSR)).

Ao receber a mensagem de *withdraw* do *Next-Hop* para a FEC, o LSR remove do registro de LSP para a FEC 2 o label *downstream* Y, e envia uma mensagem de *release* para o *Next-Hop*. O envio da mensagem de *release* é representado pelo envio do sinal *LDPReleaseDown* do processo *StateMachine* para o ambiente com os parâmetros (2=FEC, Y=label *downstream* liberado, 1111=*peer* que enviou a mensagem (LSR), 4444=*peer* que receberá a mensagem (*Next-Hop* para a FEC 2)).

Em seguida o LSR propaga a mensagem de *withdraw* para o *peer upstream*. A propagação da mensagem de *withdraw* é representada através do envio do sinal *LDPWithDrawUp* do processo *StateMachine* para o ambiente externo, figura B.1, com os parâmetros (2=FEC, B=label *upstream*, 1111=*peer* que enviou a mensagem (LSR), 123=*peer* que receberá a mensagem (*Peer UpStream*).

A situação das variáveis após a ocorrência do evento de número 7 é mostrado na tabela B.2, coluna do estado das variáveis :

- na tabela *LSPCB*, o oitavo campo do segundo registro é modificado de Y para branco.

- na tabela *Message*, o quarto registro que armazenava informações sobre o *mapping* enviado pelo *Next-Hop* 4444 para o LSR 1111 com o label Y, é removida. Esta mensagem é removida pois o LSR não se utilizará mais do label Y. Foi armazenado informações no terceiro registro, sobre a mensagem de *release* enviada de 1111 para 4444 com o label Y, indicando que o LSR não precisa mais deste label. No quarto registro, foi armazenado informações sobre a mensagem de *withdraw* propagada de 1111 para 123, com o pedido de remoção do label *upstream B*.

Oitavo Evento (*LDPRelease 2 B 123 1111*)O oitavo evento, tabela B.2, é representado pelo envio do sinal *LDPRelease* do ambiente externo ao processo *StateMachine*, com os parâmetros (2=FEC, B=label *upstream*, 123=*peer* que enviou a mensagem (*Peer UpStream*), 1111=*peer* que recebeu a mensagem (LSR)).

Ao receber a mensagem de *release* do label *upstream* para a FEC 2, label B, significa que o *peer upstream* não precisa mais utilizar o label B e, que já o removeu do registro do LSP. Desta forma, O LSR apaga o registro de LSP para a FEC 2 da tabela *LSPCB*.

Além disso, a mensagem de *withdraw* que havia sido enviada de 1111 para 123 com o label B e, que estava pendente, é removida da tabela *Message* pois, com o recebimento do *release* para o label B, o pedido de *withdraw* foi atendido. A mensagem de *mapping* que havia sido enviada de 1111 para 123 com o label B, também é removida da tabela *Message*.