UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"

Pós-Graduação em Ciência da Computação

ADRIANA FELIX ROBERTO ÁRTICO

Implementação do protocolo LMP de gerência de enlace em redes GMPLS ópticas.

Unesp

2011

ADRIANA FELIX ROBERTO ÁRTICO

Implementação do protocolo LMP de gerência de enlace em redes GMPLS ópticas.

Orientador: Prof. Dr. Mário Luiz Tronco

Dissertação de Mestrado elaborada junto ao Programa de Pós-Graduação em Ciência da Computação – Área de Concentração em Sistemas de Computação, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Unesp

2011

ADRIANA FELIX ROBERTO ÁRTICO

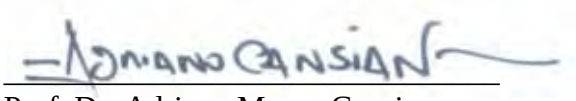
Implementação do protocolo LMP de gerência de enlace em redes GMPLS ópticas.

Dissertação apresentanda para obtenção do título de Mestre em Ciência da Computação, área de Sistemas de Computação junto ao Programa de Pós-Graduação Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

BANCA EXAMINADORA

Prof. Dr. Mário Luiz Tronco
Professor Assistente Doutor
UNESP – São José do Rio Preto
Orientador

Prof. Dr. Edilson Reis Rodrigues Kato
Professor Adjunto Doutor
Universidade Federal de São Carlos



Prof. Dr. Adriano Mauro Cansian
Professor Adjunto Doutor
UNESP – São José do Rio Preto

São José do Rio Preto, 18 de fevereiro de 2011

AGRADECIMENTOS

Agradeço primeiramente a Deus pela graça alcançada;
aos meus pais e avós pelo suporte e amor;
ao Prof. Dr. Mário Luiz Tronco pelo tempo, disposição e paciência em orientar;
ao IBILCE e seus funcionários Esther, Getúlio, Marcos e Olga, hoje colegas de profissão;
aos professores deste Instituto que me apoiaram e incentivaram ;
a Alexandre, Camila, Carlos, Fran, Jaqueline, Rafael e Thais pela amizade e enorme ajuda;
e finalmente ao meu esposo Frederico, pois ele me compreendeu, me incentivou e não me deixou desistir desta etapa tão difícil de nossas vidas.

É a você, e ao nosso meninão que está chegando, que devo esta luta. *Amo vocês.*

ÍNDICE

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AAA : Autenticação Autorização e Programação
ADM : Add-Drop Multiplexor
ASTB: Application Specific Topology Builder
ASTDL : Application Specific Topology Definition Language
BGP : Border Gateway Protocol
CC : Control Channel
CR-LDP : Constraint-based Label Distribution Protocol Overview
CSA : Client System Agent
DRAGON : Dynamic Resource Allocation in GMPLS Optical Networks
DWDM : Dense Wavelength Division Multiplexing
FSC : Fiber-Switch Capable
GMPLS : Generalized MPLS
IEEE : Institute of Electrical and Electronics Engineers
IETF: Internet Engineering Task Force
IGP: Interior Gateway Protocol
IP: Protocolo Internet
IS : Sistema Intermediário
IS-IS : Intermediate System to Intermediate System
ISCD : Interface Switching Capability Descriptor
KOM-RSVP : Multimedia Communications Lab - Resource Reservation Protocol
L2SC : Layer-2 Switch Capable
LDP : Label Distribution Protocol
LMP : Link Management Protocol
LPO : Lightpath Objects
LSC : Lambda Switch Capable
LSP : Label Switching Path
LSR :Label Switching Routers
MAC : Media Access Control
MPLS : Multi-Protocol Label Switching
NARB : Network Aware Resource Broker
NAT : Network Address Translation
OIL : Laboratory of Optical Internet
OSI: Open Systems Interconnection
OSPF-TE : Open Shortest Path First – Traffic Engineering
OXC : Optical cross-Connects
OIF : Fórum Optical Internetworking
PSC : Packet Switch Capable
PXC : Photonic cross-Connects
QoS : Quality of service
RFC : Request for Comments
ROADM : Reconfigurable Optical Add and Drop Multiplexer
RSVP: Resource Reservation Protocol (Protocolo de Reserva de Recurso)
SDH : Synchronous Digital Hierarchy
SONET : Synchronous Optical Network
TCP: Transport Control Protocol
TDM: Time-Division Multiplexing
TE: Traffic Engineering
TM : Terminal Multiplexer
TLV : Type Length Value
UCLP : User Controlled LightPath
UDP : Border Gateway Protocol
UML : Unified Modeling Language
VCI : Virtual Channel Identifier
VLSR : Virtual LSR
VNE : Virtual Network Experiment
VPI : Virtual Path Identifier
WDM : Wavelength Division Multiplexing

RESUMO

O presente trabalho descreve a implementação *open-source* do Protocolo de Gerência de Enlace denominado LMP (Link Management Protocol), baseado na RFC 4204. Este protocolo faz parte da arquitetura GMPLS (*Generalized Multi-Protocol Label Switching*), responsável por gerenciar de forma automatizada as redes ópticas.

O protocolo LMP, quando utilizado em conjunto com o pacote *open-source* DRAGON, disponibiliza ferramentas de gerenciamento distribuído apropriadas para uso em ambientes GMPLS. Neste contexto, ferramentas de roteamento distribuído, estabelecimento rápido de conexões por meio de sinalização e descoberta de serviços podem ser utilizadas. A integração do protocolo LMP *open source*, no contexto do pacote DRAGON, em equipamentos denominados ROADM (Multiplexador de Inserção/Remoção Óptico Reconfigurável) representa uma plataforma completa para o gerenciamento automatizado de redes ópticas.

Embora existam iniciativas de implementação de equipamentos ROADM nacionais, o protocolo LMP para tais plataformas, na forma *open source*, ainda não foi disponibilizado. Neste sentido, o presente trabalho traz uma contribuição para tal integração, disponibilizando o referido protocolo, para uso com o pacote DRAGON.

Foram realizados testes no Laboratório LACE com o protocolo LMP inserido no pacote DRAGON. Os resultados experimentais obtidos comprovaram as funcionalidades do protocolo relacionadas ao estabelecimento, manutenção e gerenciamento do canal de controle, além da identificação das propriedades dos nós adjacentes e isolação de falhas simples ou múltiplas no domínio óptico do gerenciamento do enlace. Tais características, são essenciais para o funcionamento completo do gerenciamento GMPLS, o qual permite a automação total de plataformas de redes ópticas.

ABSTRACT

This dissertation describes the open-source implementation of the Link Management Protocol denominated LMP (Link Management Protocol), based on RFC 4204. This protocol is part of the GMPLS (Generalized Multi-Protocol Label Switching) architecture responsible for managing automatable fiber-optic netwoks.

The Protocol LMP, when used together with the package open-source DRAGON, provides tools of management distributed appropriate for use in environments GMPLS. In this context, tools of distributed routing, fast connection establishment via signalling and service discovery can be used. The integration of open source protocol LMP, in the context of package DRAGON, in called equipment ROADM (Reconfigurable Optical Add and Drop Multiplexer) represents a complete platform for the automatized management of optical networks.

Although there are initiatives for the implementation of national ROADM equipment, the LMP protocol for such platforms, in the form open source, has not yet been made available. In this direction, the present work brings a contribution for such integration, make available the related protocol, for use with package DRAGON.

Tests were performed in the Laboratory LACE with the LMP protocol in package DRAGON. The results experimental had proven the related functionalities of the protocol to the control channel establishment, maintenance and management, beyond the identification the properties of adjacent nodes and isolation of single or multiple failures in the optical domain management link. Such characteristics, are essential for the complete functioning of management GMPLS, which allows the total automation of platforms of optical networks.

1. INTRODUÇÃO

Nos últimos anos houve um grande crescimento da internet, aumentando, assim, o volume de tráfego de informações. Acompanhando este crescimento, surgiu a necessidade de confiança, eficiência e qualidade de serviço em relação às redes utilizadas.

Para suprir algumas das necessidades desta evolução, tanto as redes locais como as metropolitanas adotaram o padrão *Ethernet*. Sua principal vantagem está na simplicidade e baixo custo por *bits* proporcionados, além de ser mais rápido para grandes transportes de dados. Esse padrão, porém, não oferece qualidade de serviço. Nesse contexto, a engenharia de tráfego emerge com maior importância no desenho e operação de grandes redes.

Redes ópticas gerenciáveis apresentam-se como uma tecnologia capaze de suportar todos os requisitos de serviço e de qualidade resultantes desse crescimento, em volume de tráfego, tamanho geográfico e diversificação de aplicações. Pesquisas prevêem um grande aumento da capacidade de transporte e da escalabilidade das redes atuais, além do desenvolvimento de uma grande quantidade de novas e sofisticadas aplicações. Desta forma, as redes ópticas baseadas na multiplexação por divisão de comprimento de onda e na comutação puramente óptica se mostram como candidatas ideais, uma vez que superam as limitações das infra-estruturas atuais (baseadas em comutação eletrônica) e permitem taxas de transmissão da ordem de algumas dezenas de *terabits* por segundo, valor próximo ao limiar teórico das fibras ópticas.

As soluções até agora apresentadas para tal crescimento dependem de extensões que precisam ser realizadas, principalmente, nos protocolos de roteamento entre domínios, para suportar a distribuição de informações de engenharia de tráfego. Como uma alternativa em relação aos processos de padronização que propõem extensões e uma maior integração da camada IP (*Internet Protocol*) com a camada ótica, a tecnologia MPLS (*Multi-protocol Label Switching)* foi desenvolvida criando uma camada de serviços que facilita a interação entre diferentes domínios administrativos (CUNHA, 2006).

Para se ter uma rede verdadeiramente dinâmica, é necessário um método para controle total da rede dentro do núcleo óptico. Surgiu, então, o conceito de rede óptica “inteligente”. O

MPLS pode ser utilizado como rede inteligente, mas, como é específico para redes IP e não para ópticas, seus protocolos foram modificados para "conversar" com equipamentos de telecomunicação, propiciando, desta forma, o surgimento do MPLS Generalizado (GMPLS – *Generalized MPLS*).

Os protocolos definidos para a arquitetura MPLS suportam somente a comutação de pacotes. No entanto, o GMPLS atende comutação de *slots* de tempo (*Time Division Multiplexing - TDM*), comutação por comprimento de onda (*Wavelength Division Multiplexing - WDM*) e comutação por fibra (CHEGE & FREELANCE, 2009).

O GMPLS, extensão do paradigma MPLS às redes ópticas, é um conjunto de protocolos do plano de controle que fornece uma semântica consistente e uniforme para sinalização (*Resource Reservations Protocol – Traffic Engineering* – RSVP-TE), roteamento (*Open Shortest Path First – Traffic Engineering* – OSPF-TE) e gerenciamento de enlace (*Link Management Protocol* – LMP).

Um par de nós de uma rede pode ter milhares de interconexões, cada uma podendo consistir de múltiplos enlaces de dados quando multiplexados (por exemplo, *slots* TDM ou comprimento de ondas em WDM). Neste contexto, a IETF (*Internet Engineering Task Force*) padronizou um protocolo denominado LMP que atua entre um par de nós, para ser usado no gerenciamento dos enlaces e para verificar a alcançabilidade do canal de controle.

O projeto DRAGON (*Dynamic Resource Allocation via GMPLS Optical Networks*) é uma parceria entre um conjunto de instituições de pesquisas americanas para promover uma infra-estrutura que permita às aplicações de redes adquirirem recursos de forma dinâmica e determinística, e integrá-los aos diversos dispositivos que formam a rede. Esse protótipo faz uso de um plano de controle baseado no GMPLS, e tem como base atender alguns requisitos como: roteamento distribuído, estabelecimento rápido de conexão por meio de sinalização e descoberta de serviços. A funcionalidade de gerenciamento dos enlaces (baseado no protocolo LMP) é o único protocolo do plano de controle GMPLS que não foi implementado nesse projeto.

O LMP oferece o gerenciamento dos enlaces, sendo responsável pela criação de *links* virtuais entre dois nós distintos no plano de dados e pela gerência destes *links* no plano de controle, verificando a qualidade das conectividades físicas entre os nós vizinhos, a identificação das propriedades dos nós adjacentes e a isolação de falhas simples ou múltiplas no domínio óptico (SOUZA et al., 2005). Desta forma, o LMP proporciona o estabelecimento, a manutenção e o gerenciamento do canal de controle, sendo essencial para garantir as funcionalidades do GMPLS acima citadas.

1.1. Motivações

Com a finalidade de explorar a capacidade total das fibras ópticas, os quais trabalham na ordem de dezenas de *Terabits* por segundo (Tbps), as redes ópticas atuais têm sido desenvolvidas com equipamentos como: roteadores, *switches,* comutadores fotônicos (*Photonic Cross-Connects* – PXC), comutadores ópticos (*Optical Cross-Connects* – OXC), sistemas DWDM (*Dense Wavelength Division Multiplexing*) e multiplexadores *add-drop* (ADMs). Tais equipamentos, na maioria das aplicações, utilizam o GMPLS como um plano de controle comum para, dinamicamente, alocar recursos e manter a rede operacional.

A motivação para este trabalho tem origem no projeto GIGA, uma iniciativa nacional de implantação e operação de uma Rede Experimental de Alta Velocidade (BARROS et al., 2005). O objetivo de tal projeto é obter novas soluções para redes IP/WDM, desenvolvendo protótipos de *hardware* e *software.* As funcionalidades providas neste contexto envolvem: capacidade, alcance geográfico, integração, flexibilidade, reconfigurabilidade, confiabilidade e robustez frente a falhas.

Dentre os equipamentos (*hardwares*) desenvolvidos pelo projeto GIGA se encontram o multiplexador de inserção/ remoção que utiliza a arquitetura GMPLS, este é conhecido como ROADM (Multiplexador de Inserção/Remoção Óptico Reconfigurável) e permite que a rede óptica seja totalmente automatizada. Esta tecnologia pode adicionar/remover canais ópticos em um nó. Um Eles possuem baixo custo de fabricação, podem ser controlados remotamente, são reconfiguráveis e aceitam balanceamento automático de carga. A atribuição de largura de banda nestes equipamentos não precisa ser realizada durante a implantação inicial, e sim, quando necessária (VELASCO et al., 2007). A desvantagem destes equipamentos disponíveis é que são importados, de alto custo.

Embora sejam grandes os esforços para a disponibilização de soluções de *software* que contemplem o conjunto de protocolos necessário para a integração do *hardware* desenvolvido, o protocolo LMP, de gerência de enlace, e outros protocolos do plano de controle GMPLS não estão sendo desenvolvidos como versão *open-source*.

As principais motivações para esta dissertação baseiam-se no estudo da solução GMPLS com ênfase no protocolo LMP e tem como finalidade disponibilizar uma versão *open-source* desse protocolo, integrado ao pacote DRAGON provendo, assim, uma ferramenta completa de gerenciamento GMPLS, a qual poderá ser integrada aos equipamentos de diversos tipos e fabricantes em curto prazo, apresentando-se como uma solução nacional com preços mais acessíveis.

1.2. Principais Objetivos

Dentro do contexto mostrado anteriormente, os objetivos do presente trabalho envolvem:

• Entender o funcionamento e a finalidade de uma arquitetura para aprovisionamento e gerência de serviços em redes ópticas, com ênfase no protocolo LMP.

• Acrescentar ao pacote DRAGON a implementação do protocolo LMP em versão *open-source*, com a finalidade de garantir os aspectos relacionados à qualidade de serviço em redes GMPLS.

1.3. Organização

O segundo capítulo apresenta análises teóricas da arquitetura GMPLS em redes ópticas e do protocolo LMP, que será utilizado na implementação do algoritmo proposto. O terceiro capítulo aborda o projeto que envolve esse trabalho, apresentando o pacote DRAGON e seus respectivos protocolos. Por sua vez, o quarto capítulo mostra os testes realizados e descreve o desenvolvimento e a implementação do trabalho. A conclusão discute as principais contribuições obtidas no presente trabalho.

2. PROTOCOLO LMP E ARQUITETURA GMPLS

Neste capítulo são apresentados os principais conceitos relacionados à arquitetura GMPLS (*Generalized Multi-protocol Label Switching)*, detalhando a pilha de protocolos capaz de prover uma rede óptica inteligente e dinâmica que controla o estabelecimento e manutenção de conexões. Um destes protocolos, responsável pelo gerenciamento do canal de controle dessa arquitetura e gerenciamento dos enlaces da rede, o LMP, é então apresentado e detalhado.

2.1. Arquitetura GMPLS

Com o crescimento do número de usuários nas redes e com aumento de aplicativos gráficos, multimídia e cooperativos, tanto as redes locais como as metropolitanas adotaram o padrão *Ethernet* (*Fast Ethernet* e da *Gigabit Ehernet*). Sua principal vantagem está na simplicidade e baixo custo por *bits* proporcionados, além de ser mais rápido para grandes transportes de dados. Esse padrão, porém, não oferece qualidade de serviço oferecida pela engenharia de tráfego. Nesse contexto, essa engenharia de tráfego emerge com maior importância no desenho e operação de grandes redes.

Redes GMPLS, como definidas pelo IETF (*Internet Engineering Task Force*) e pelo OIF (*Optical Internetworking Fórum*), atendem a necessidade de múltiplos tipos de tráfego e de camadas (IP/ATM/SONET-SDH) e dão suporte à interoperabilidade de equipamentos e diferentes fabricantes. Tais redes exploram os limites de transmissão na fibra óptica, podendo trabalhar a dezenas e até centenas de *Gbps* de banda disponível, atendendo a todos os tipos de aplicações que requerem altas taxas de transmissão.

As redes ópticas consistem basicamente de roteadores, *switches*, multiplexadores OAD *(Add-Drop Multiplexor*), comutadores fotônicos (PXC - *Photonic cross-Connects*), comutadores ópticos (OXC - *Optical cross-Connects*) e fazem uso de técnicas de multiplexação, DWDM (*Dense Wavelength Division Multiplexing*) (VERDI, 2006).

A partir destas tecnologias, arquiteturas de redes ópticas inteligentes vêm sendo definidas. Seus maiores benefícios são o aprovisionamento de conexões de forma automática e dinâmica, a descoberta automática de topologias e recursos, a engenharia de tráfego para a otimização dos recursos de rede e a capacidade automática de proteção e restauração de falhas.

Nas redes ópticas, normalmente, existe a separação do plano de controle e do plano de transporte. Neste caso, as funcionalidades de controle não estão implementadas nos elementos de rede, mas sim, em agentes de controle que se comunicam via canais de controle próprios ou via rede de controle separada, na qual um único agente de controle pode representar múltiplos elementos de rede (FEDYK et al., 2010).

O plano de controle refere-se à infra-estrutura e inteligência distribuída que controla o estabelecimento e a manutenção de conexões na rede. A arquitetura GMPLS propõe que esta inteligência seja realizada por meio de um conjunto de vários protocolos de comunicação para o aprovisionamento automático de conexões ópticas. Tais protocolos incluem protocolos de sinalização, roteamento e descoberta. Essa arquitetura estende os protocolos definidos para o MPLS, a fim de atender aos novos tipos de tecnologia de comutação de rótulos (CRISPIM, 2006).

Diferentemente das redes IP, nas quais os pacotes são encaminhados com base no endereço IP de destino (contido no cabeçalho deste pacote) e na informação da base de dados do roteador, o MPLS tem como apoio para o encaminhamento do pacote os rótulos atribuídos a cada pacote (*Label Switching*), que possui uma base de dados de envio, e o caminho utilizado pelos pacotes rotulados são chamados de *Label Switching Path* (LSPs). Esses rótulos têm comprimento fixo e pequeno, agilizando o roteamento dos pacotes. O MPLS tem a finalidade de comutar multi-camadas, ou seja, pode ser aplicado a diferentes tecnologias de comutação de pacotes, células ou *frames* (IP, ATM, *Frame Relay*, etc.).

O GMPLS (*Generalized Multi-Protocol Label Switching*) tem como objetivo encontrar e providenciar um caminho (LSP) ótimo para os pedidos de tráfego de um usuário. Sua arquitetura, apresentada na Figura 1, mostra os protocolos de sinalização RSVP-TE (*Resource Reservation Protocol – Traffic Engineering*) e CR-LDP (*Constraint-based Label Distribution Protocol Overview*), os protocolos de roteamento OSPF-TE (*Open Shortest Path First – Traffic Engineering*), IS/IS (*Intermediate System to Intermediate System*) e BGP (*Border Gateway Protocol*) (com outro Sistema Autônomo), o protocolo de gerência de enlace LMP (*Link Management Protocol*), os protocolos de transporte UDP (*Border Gateway Protocol*) e TCP (*Transport Control Protocol*), e o protocolo de rede IP (*Internet Protocol*).

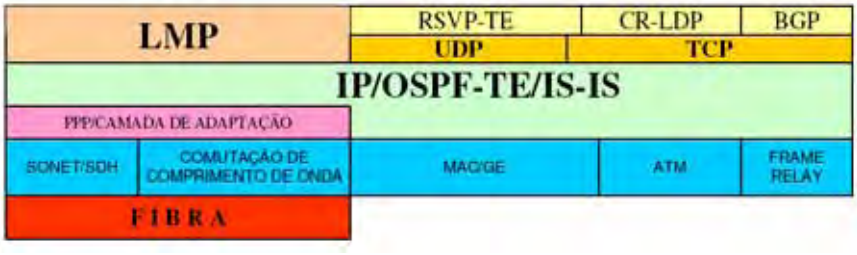


Figura 1 – Pilha de Protocolos GMPLS (BRASSOLATI, 2006).

Em azul, na Figura 1, são apresentados os tipos de comutação do GMPLS, que foram estendidos do MPLS, para prover um plano de controle (sinalização e roteamento), não somente para os dispositivos que executam a comutação de pacotes, mas também a aqueles que executam a comutação em *slots* de tempo, comprimentos de onda e fibras. Estes dispositivos são roteadores (*Label Switching Routers* – LSRs) com um conjunto de interfaces que executam outras operações de comutação, além da comutação de pacotes.

As interfaces podem ser classificadas como (PAPADIMITRIOU et al., 2006):

• Interfaces PSC (*Packet Switch Capable*): são interfaces que recebem pacotes de entrada e encaminham os dados baseados no conteúdo (conhecido como rótulo) do cabeçalho do pacote. Exemplo: roteadores MPLS.

• Interfaces L2SC (*Layer-2 Switch Capable*): recebem quadros/células e comutam os dados baseados no conteúdo do cabeçalho do quadro/célula. Exemplo: interfaces em pontes *Ethernet* que comutam dados baseados no conteúdo do cabeçalho MAC.

• Interfaces TDM (*Time-Division Multiplex Capable*): comutam dados baseados nos *slots* de tempo em um ciclo de repetição. Exemplos: interfaces em comutadores SONET/SDH (SONET/SDH *Cross-Connect*), multiplexadores (*Terminal Multiplexer* – TM) e multiplexadores *Add-Drop* (*Add-Drop Multiplexer* – ADM).

• Interfaces LSC (*Lambda Switch Capable*): permutam dados com base no comprimento de onda em que o dado é recebido. Exemplos: comutadores fotônicos (PXCs) e comutadores ópticos (OXCs).

• Interfaces FSC (*Fiber-Switch Capable*): são interfaces que permutam dados, considerando a posição dos dados no espaço físico (fibra, porta).

Para atender a todas essas interfaces, o GMPLS faz uso de algumas inovações tecnológicas, entre as quais a hierarquia LSP (*Label Switch Path*), na qual diferentes interfaces de controle são colocadas dentro de LSPs (LSP dentro de outra LSP), com um

mesmo tipo de interface ou com um tipo diferente. A ordem dessa hierarquia para diferentes interfaces é mostrada na Figura 2.

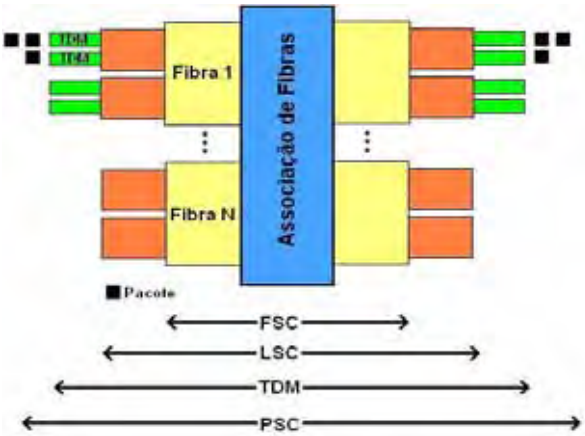


Figura 2 – Hierarquia LSP (CUNHA, 2006).

No topo da hierarquia, estão as interfaces FSCs, seguidas pelas interfaces LSCs, TDMs e finalizando com as interfaces PSCs. A interface L2SC, não citada na figura, encontra-se entre as TDM e PSC. Dessa forma, um LSP que começa e termina com uma interface PSC pode ser encaixado em um LSP que começa e termina com uma interface TDM. Por outro lado, esse LSP pode ser encaixado em uma LSP que começa e termina com uma interface LSC, que, por sua vez, pode ser encaixado em um LSP que começa e termina com uma interface FSC.

Como os nós intermediários vêem somente o LSP externo, eles não precisam manter os estados de envio para cada LSP interno, diminuindo o número de mensagens de sinalização e aumentando, assim, a escalabilidade de sinalização.

O GMPLS é composto pelo plano de transferência, contendo o fluxo de tráfego dos dados de transferência, e pelo plano do controle, com o fluxo do pacote de controle. O plano de transferência é dividido em: camada do pacote (PSC) e camada ótica (LSC), que colabora com a próxima camada, por meio do plano do controle.

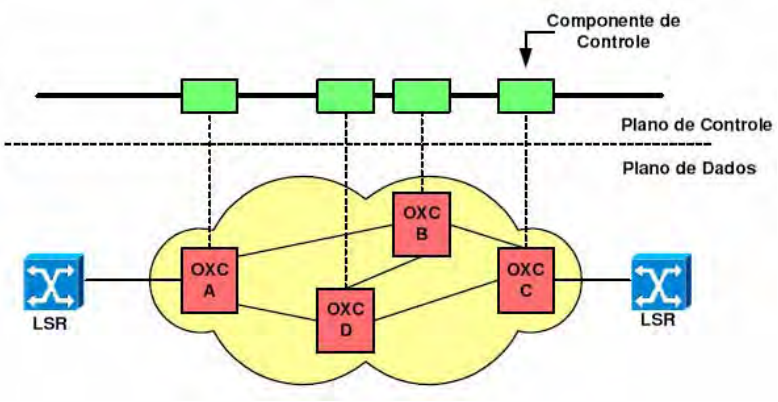


Figura 3 – Funcionamento de uma Rede GMPLS (CUNHA, 2006).

Como mostrado na Figura 3, para a colaboração de camadas múltiplas, um componente de controle é integrado na rede, ajustando o caminho na camada ótica, para que concorde com o tráfego do IP na camada do pacote e na recuperação rápida de falhas (TAGAME et al., 2007).

Uma das principais características do MPLS e, consequentemente, do GMPLS é a engenharia de tráfego (TE). Com os custos elevados de redes e dos equipamentos ópticos, a TE transformou-se numa função indispensável dentro de muitos domínios de transporte. Como definido, a TE está preocupada com os recursos e com a atuação da otimização da rede no seu desempenho, incluindo a aplicação de princípios científicos de medida, modelagem, caracterização e controle de tráfego. Esta aplicação da engenharia corresponde às técnicas cujo objetivo é alcançar a eficiência e a confiança das operações de redes ópticas, usando, simultaneamente, o recurso de otimização da rede e o de desempenho do tráfego. Aspectos da TE que são de interesse para o tráfego das redes ópticas projetadas envolvem medida e controle.

Com base nessa engenharia, outra inovação tecnológica do GMPLS é o estabelecimento de um LSP bidirecional, que melhora o tempo de aprovisionamento da conexão. Ao usar um MPLS, considerando a TE, estabelecer uma LSP bidirecional significa utilizar duas LSPs unidirecionais independentes. Dentro do contexto de GMPLS, o estabelecimento simultâneo de ambos os trajetos, *downstream* (contra o fluxo de dados) e *upstream* (a favor ao fluxo de dados), usa um único conjunto de mensagens de sinalização. Para os LSPs bidirecionais, dois rótulos devem ser alocados: um para o trajeto *downstream* (pedido pelo nó *upstream* e atribuído pelo nó *downstream*) e outro para o trajeto *upstream* (RUBUYAT, 2006).

As principais vantagens do GMPLS envolvem a redução da complexidade de projeto (desenvolvimento e manutenção), descobrimento/controle de recursos distribuídos em tempo real; recuperação/estabelecimento de conexão de múltiplas camadas; aprovisionamento de circuito rápido e flexibilidade de serviço. Como exemplo pode-se citar serviços de largura de banda sob demanda, melhor interoperabilidade dos elementos de rede de diferentes fabricantes e aumento da capacidade de sobrevivência, fornecendo a habilidade de re-roteamento dinâmico quando uma falha ocorre, facilitando, assim, as operações do dia-a-dia das operadoras (CUNHA, 2006).

Hoje, baseado em exigências de transporte, a maioria dos esforços de GMPLS são dirigidos para as extensões dos protocolos de IP/MPLS, que controlam as redes, incluindo as redes óticas. Esse conjunto de protocolos é definido para o plano de controle, conforme exposto na Figura 4.

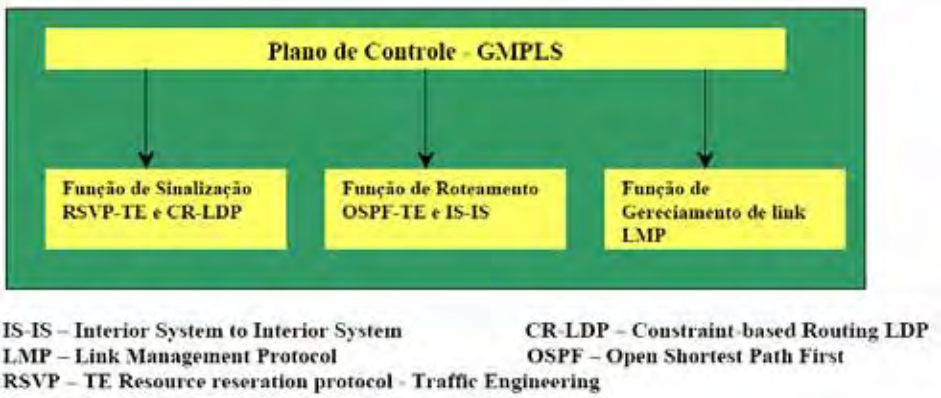


Figura 4 – Plano de Controle GMPLS (ALOIA, 2009).

O plano de controle GMPLS possui três funções: sinalização, roteamento e gerenciamento de enlaces. A função dos protocolos de roteamento envolve a descoberta de topologia da rede (descoberta de vizinhos), deixando a rede pronta para aceitar os pedidos de serviço. Já os protocolos de sinalização têm o objetivo de garantir que todos os serviços tenham suas QoS atendidas, reservando os recursos ao longo do caminho.

Os protocolos de sinalização e roteamento requerem, ao menos, um canal de controle bidirecional para diagnosticar se dois LSR adjacentes estão conectados mediante enlaces unidirecionais. O LMP pode estabelecer, manter e gerenciar estes canais de controle. Os conjuntos de protocolos serão descritos mais detalhadamente nas próximas seções.

2.1.1. Extensões em Relação ao MPLS

Para aceitar dispositivos com novas formas de comutação, além da comutação por pacotes, o GMPLS encontrou a necessidade de acrescentar uma nova complexidade na rede, requerendo soluções para os problemas de roteamento e sinalização, antes não encontrados.

Conforme a RFC 3945 (MANNIE, 2004), algumas padronizações da arquitetura GMPLS mostram as extensões dos protocolos de roteamento (IS-IS ou OSPF) e protocolos de sinalização (RSVP ou CR-LDP), além do protocolo LMP. As principais extensões dos protocolos em relação ao MPLS serão apresentadas nas próximas seções.

Protocolos de roteamento

Com os objetivos de descoberta da rede e disseminação da métrica, os dois protocolos descritos por Adeel (ADEEL et al., 2005) são indicados para a função de roteamento em GMPLS: o OSPF (*Open Short Path First*) e o IS-IS (*Intermediate System to Intermediate System*).

O OSPF é utilizado no interior de sistemas autônomos (*Interior Gateway Protocol* – IGP) para a troca de informações de rotas dos pacotes IP. A métrica usada para computar a tabela de roteamento é o *link-state*, isto é, os roteadores que executam esse protocolo trocam entre si informações sobre os estados dos enlaces de comunicação ligados às suas portas. Cada roteador na rede torna-se responsável por descobrir seus roteadores vizinhos, e, então, anunciar identidades aos seus vizinhos adjacentes e o custo para alcançar cada um deles. As características que merecem destaque são: a fácil extensão para engenharia de tráfrgo, informação do *wavelength* e escalabilidade.

O IS-IS, na terminologia ISO, em que um roteador é consultado como Sistema Intermediário (IS), também é um protocolo do interior de sistemas autônomos para o roteamento de redes IP, baseado no caminho mais curto - SPF (*Short Path First*) e no algoritmo do *link-state*. O IS-IS tem muitas características desejáveis, incluindo: escalabilidade para redes maiores (baixo controle da rede), *Forwarding Multi-Path* e comunicação múltipla. O IS-IS não é um padrão IETF e é definido em RFCs somente. Como consequência, IS-IS não é um protocolo aberto do IGP, portanto, não é uma escolha popular para a extensão ao domínio óptico.

Extensões dos protocolos de roteamento

As extensões introduzidas nos protocolos de roteamento têm o objetivo de autodescoberta e propagação da topologia da rede, alcançabilidade, anúncio da disponibilidade de recursos (por ex.: largura de banda e tipo de proteção) e capacidade da rede via plano de controle. De acordo com a RFC 4202, as principais extensões são mostradas na Figura 5:

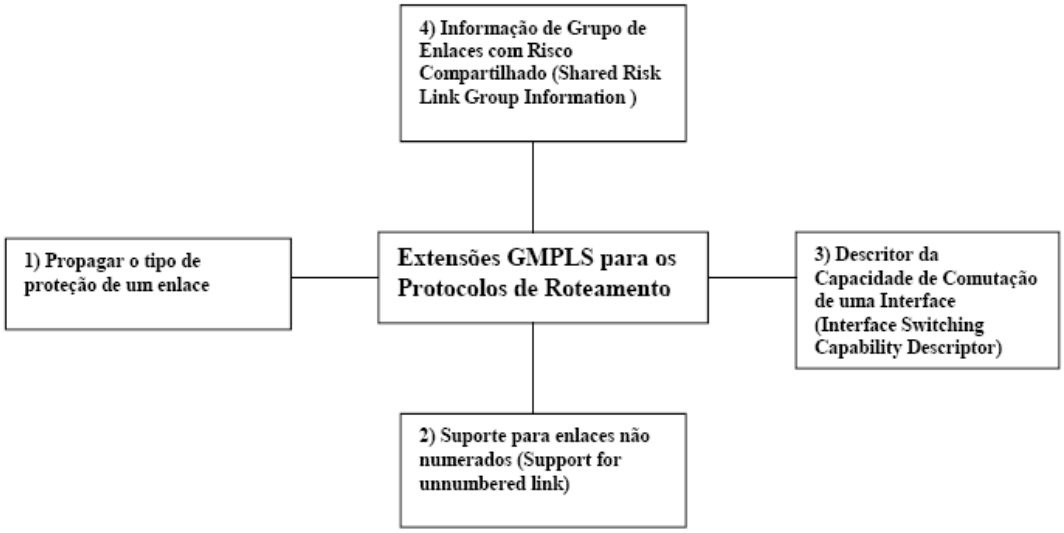


Figura 5 – Extensões GMPLS para os Protocolos de Roteamento (ALOIA, 2003).

1) Capacidade de proteção de um enlace

É desejável carregar a informação de modo que possa ser utilizada pelo algoritmo de configuração do caminho na criação de LSPs com características de proteção específicas. A informação é organizada em uma hierarquia na orientação: da menor a mais elevada proteção. Vale enfatizar que a função de informar a capacidade de proteção de um enlace é opcional.

As capacidades de proteção são definidas a seguir:

Tráfego extra (Extra Traffic): O enlace é destinado à proteção de enlace(s) primário (s), e tais recursos devem ser devolvidos caso uma falha ocorra nestes.

Sem proteção (Unprotected): Não existe nenhum enlace protegendo um determinado enlace. A LSP com um enlace deste tipo será perdida se o enlace falhar.

Proteção distribuída (Shared): Se o tipo de enlace é distribuído, significa que há um ou mais enlaces do tipo “extra tráfego” separados, protegendo este enlace. Esses enlaces do tipo “extra tráfego” estão distribuídos entre um ou mais enlaces do tipo compartilhado.

Proteção dedicada (Dedicated 1:1): Significa que há um enlace do tipo "extra tráfego" separado que está protegendo este enlace.

Proteção dedicada (Dedicated 1+1): Um enlace separado e dedicado protege este enlace. No entanto, a proteção não é anunciada no banco de dados do estado do enlace e, portanto, não está disponível para o roteamento no LSP.

Proteção estendida (Enhanced): Indica que um regime de proteção de maior confiança do que a "proteção dedicada 1+1" está sendo usado para proteger esta ligação.

2) Suporte para enlaces não-numerados (*Support For Unnumbered Links*)

Enlaces não-numerados são ligações ponto a ponto que não possuem identificação de rede (IPV4 ou IPV6). Cada um dos LSPs presentes nas extremidades de um enlace não-numerado deve se atribuir, então, um identificador de 32 *bits*. Ter o suporte a enlaces não-numerados no roteamento significa incluir informações sobre os identificadores deste enlace, que se propaga tanto nos identificadores locais como nos remotos (valor 0 caso não se identifique a ligação remota). A redução de esforços de gerenciamento de endereço IP, por meio da aplicação deste fundamento, torna-se um aspecto muito significativo, especialmente, no contexto de redes ópticas, nas quais pode existir uma grande quantidade de enlaces entre nós (ALOIA, 2003).

3) Descritor da capacidade de comutação de uma interface (*Interface Switching Capability Descriptor* – ISCD)

No contexto GMPLS, um enlace está conectado a um nó por uma interface, que pode ter diferentes capacidades de comutação. A cada interface é associado um descritor de sua capacidade de comutação (ISCD), sendo este descritor propagado pelos protocolos de roteamento e utilizado na computação de um caminho por meio da rede. Uma interface pode ter mais do que um ISCD quando há interfaces que suportam múltiplas capacidades de comutação. Por sua vez, se não houver nenhum descritor associado à interface, assume-se como sendo capaz de comutar pacotes, ou seja, a interface PSC.

4) Informação de grupo de enlaces com risco compartilhado (*Shared Risk Link Group Information* – SRLG)

A função de informar o grupo de enlaces com risco compartilhado é opcional. Um conjunto de enlaces pode constituir um "grupo de enlaces com risco compartilhado" (SRLG), que se compartilham um recurso cuja falha pode afetar todos os enlaces do conjunto. Por

exemplo: duas fibras ópticas em um mesmo duto podem estar no mesmo SRLG. Um enlace pode pertencer a vários SRLGs. Um SLRG é identificado por um número de 32 *bits*, único dentro do domínio IGP (*Interior Gateway Protocol*).

Protocolos de sinalização

Os protocolos de sinalização garantem o estabelecimento das LSPs com TE. Dentre os protocolos, o RSVP realiza a reserva de recursos da rede antes mesmo de se iniciar a transferência de dados, e o CR-LDP é composto por procedimentos e mensagens.

O RSVP-TE é um protocolo de sinalização de reserva de recursos da arquitetura MPLS, empregado para suportar serviços integrados sobre a rede. A idéia de serviços integrados é a reserva de recursos, ou seja, as aplicações devem encontrar uma rota até o receptor que satisfaça suas demandas de qualidade de serviço, reservando, ao mesmo tempo, os recursos necessários antes do início da transmissão dos dados.

O CR-LDP define um conjunto de procedimentos e mensagens pelas quais os LSRs estabelecem LSPs por meio da rede, mapeando a informação da camada de rede para caminhos comutados na camada de enlace.

Existem quatro categorias de mensagens LDP:

Mensagens de descoberta: anunciam a presença de um LSR em uma rede e confirmam se o mesmo continua presente.

Mensagens de sessão: estabelecem, mantêm e terminam sessões entre dois LSRs.

Mensagens de anúncio de rótulo: criam, modificam e suprimem associações rótulo-FEC (*Forwarding Equivalence Class*).

Mensagens de notificação: provêem informações do estado da rede e sinalizam erros.

Extensões dos protocolos de sinalização

As extensões dos protocolos de sinalização são utilizadas para determinar a maneira como os rótulos são solicitados e distribuídos, para trocar informações de sincronização, para a alocação de rótulos, entre outras funções (TOMBI, 2007).

Aloia (ALOIA, 2003) utilizou as RFCs 3472 e 3473 e referenciou dez extensões criadas para os protocolos de sinalização, conforme apresentadas na Figura 6:

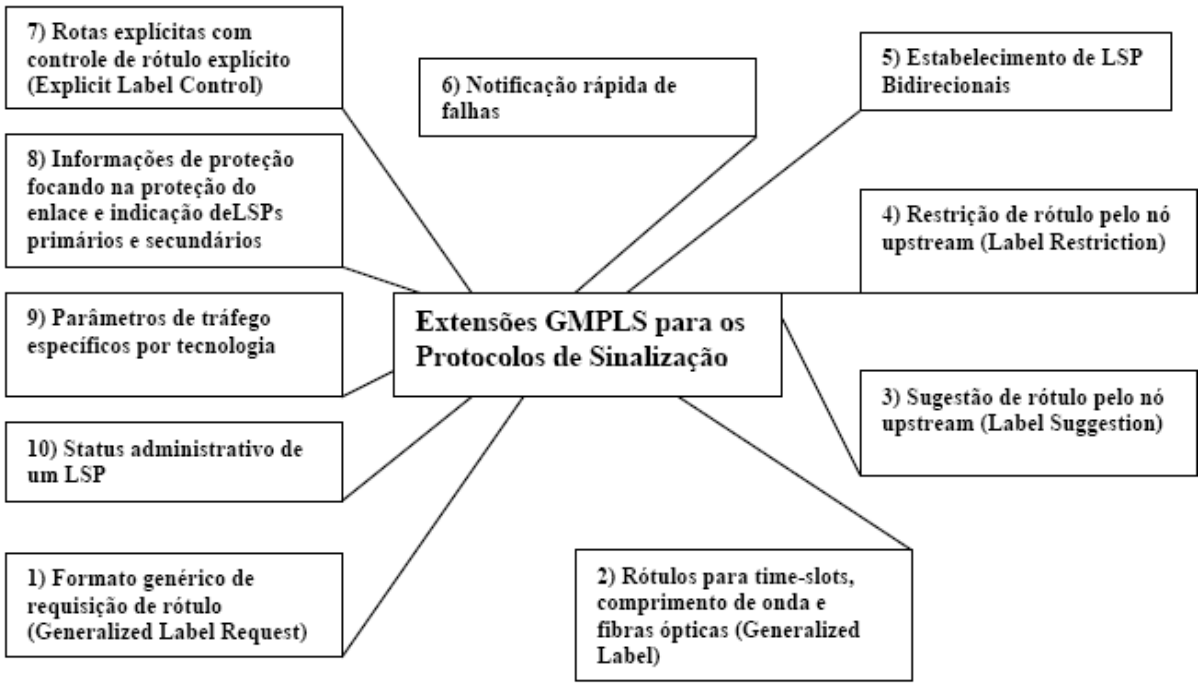


Figura 6 – Extensões GMPLS para os Protocolos de Sinalização (ALOIA, 2003).

Dentre as extensões, as funções 1, 2 e 9 são obrigatórias; as funções 5 e 7 devem ser implementadas; e as funções 3, 4, 6, 8 e 10 são opcionais. Isso proporciona diversas formas de implementar a sinalização na rede.

1) Rótulo genérico de requisição (*Generalized Label Request*)

Nas redes GMPLS, os dados contidos em fluxo requisitado não são sempre associados a um valor de rótulo, ou seja, o valor do rótulo é implícito, pelo fato dos pacotes serem transportados por comprimentos de onda específicos, por intervalos de tempo ou por fibras ópticas. Por outro lado, alguma representação do valor do rótulo é requisitada pelos protocolos de sinalização, para que as mensagens de controle entre os equipamentos possam concordar com o valor e ser utilizado. O pedido de rótulo genérico introduz um esquema de codificação. Para o protocolo de sinalização RSVP, esse esquema refere-se a um objeto denominado *Generalized Label*, e, para o protocolo CR-LDP, um *Type-Length-Value* (TLV), que carrega o rótulo e as informações associadas.

O *Generalized Label Request* apresenta três parâmetros como mostra a Figura 7.

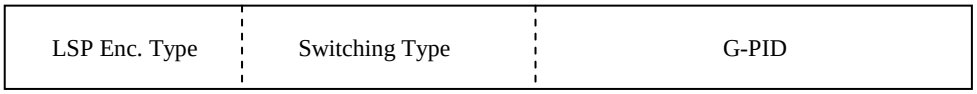


Figura 7 – Informação Transportada em um *Generalized Label Request*.

O campo *LSP Encoding Type* possui um tamanho de 8 *bits* e tem a função de indicar o tipo de codificação a ser usado pelos dados associados a um LSP. Existem alguns valores padronizados, como: 1 (Pacote), 2 (*Ethernet*), 8 (*Lambda*), 9 (Fibra). O campo *Switching Type* também possui tamanho de 8 *bits*, cuja função é indicar o tipo de comutação a ser executada em um enlace particular para um LSP. O campo G-PID (*Generalized Playload Identifier*) tem tamanho de 16 *bits* e serve para identificar a carga transportada por um LSP. Há alguns valores-padrão: 33 (*Ethernet*), 37 (comprimento de onda), 34 (SONET/SDH). Outros são parâmetros específicos não são transportados em um *Generalized Label Request*, mas sim, em parâmetros de tráfegos para tecnologias específicas.

2) Rótulo genérico (*Generalized Label*)

A função do rótulo genérico envolve a resposta a um pedido de rótulo. Um rótulo genérico somente transporta um único nível de rótulo não-hierárquico. Quando múltiplos níveis de rótulos (LSPs dentro de LSPs) são necessários, cada LSP deve ser estabelecido separadamente. Os rótulos para comprimentos de onda e portas de um comutador apresentam este formato variável, de comprimento de 32 *bits.*

3) Sugestão de rótulo (*Label Suggestion*)

A sugestão de rótulo é responsável por informar a um nó, cujo sentido esteja contra o fluxo de transmissão (*downstream*), a preferência para o nó que esteja a favor (*upstream*). Tal função é útil quando se estabelecem LSPs por meio de certos tipos de equipamentos ópticos, nos quais pode existir um atraso na configuração de sua matriz de comutação. O uso da Sugestão de Rótulo é somente um aperfeiçoamento, ou seja, sua transmissão não implica em sua utilização para a transmissão de dados.

4) Restrição de Rótulo (*Label Restriction*)

Um nó *upstream* pode limitar a escolha de rótulos de um nó *downstream* em um conjunto de rótulos aceitáveis, enviando-lhe listas ou faixas de rótulos aceitáveis, obedecendo algumas restrições. Há três casos em que a restrição de rótulo no domínio óptico é útil:

• onde um equipamento final é capaz de transmitir e receber somente em um pequeno conjunto de comprimentos de onda;

• onde existe uma sequência de interfaces que não suportam conversão de comprimentos de onda e requerem o uso do mesmo comprimento de onda fim-a-fim em uma sequência de nós por todo o caminho;

• onde dois equipamentos finais de um enlace suportam diferentes conjuntos de comprimento de onda.

O nó destinatário de um conjunto de rótulos deve restringir sua escolha a este conjunto. Um conjunto de rótulos pode estar presente por meio de múltiplos nós. Em cada um destes, será gerado seu próprio conjunto de rótulos, baseado no conjunto de rótulos enviado pelo nó anterior, e seus próprios recursos de *hardware.*

A Figura 8 mostra o formato de um conjunto de rótulos e seus componentes.

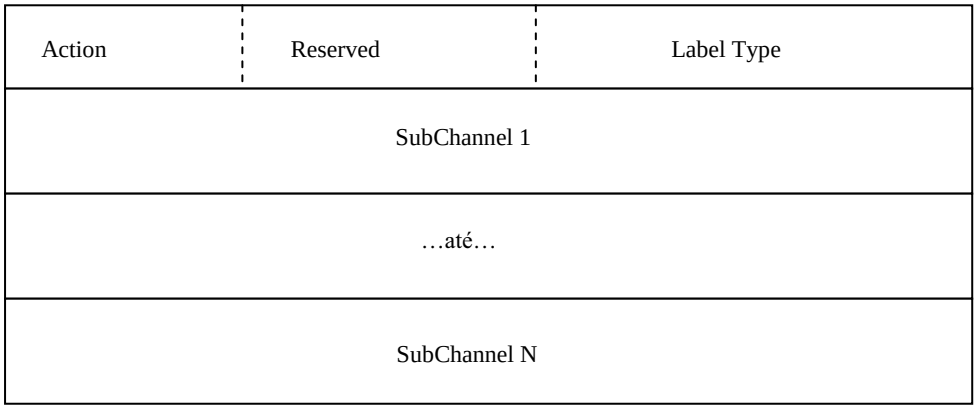


Figura 8 – Formato de um Conjunto de Rótulos e os Componentes.

O campo *Action* possui tamanho de 8 *bits.* Quando tem valor 0, indica que um objeto (RSVP)/TLV (*Value Label Type*) (CR-LDP) contém um ou mais subcanais incluídos em um conjunto de rótulos. Com valor 1, indica que um objeto (RSVP)/TLV (CR-LDP) contém um ou mais subcanais excluídos em um conjunto de rótulos. Quando o campo tem valor 2, um objeto (RSVP)/TLV (CR-LDP) contém uma faixa de rótulos, sendo o primeiro subcanal a origem e o segundo o fim da faixa. O valor 3 indica que um objeto (RSVP)/TLV (CR-LDP) contém uma faixa de rótulos excluída do conjunto de rótulos, sendo o primeiro subcanal a origem e o segundo o fim da faixa. O campo *Reserved* tem tamanho de 10 *bits.* O campo *Label Type* possui tamanho de 14 *bits*, e sua função é indicar o tipo e o formato de um rótulo carregado em um objeto (RSVP)/TLV (CR-LDP). O subcanal representa o rótulo.

5) Estabelecimento de LSP bidirecional

Na arquitetura GMPLS, as LPSs são unidirecionais. A fim de estabelecer um LSP bidirecional, dois LSPs unidirecionais em direções opostas devem ser estabelecidos independentemente. Tais LSPs apresentam duas desvantagens: a latência para estabelecer dois LSPs é alta; o *overhead* é o dobro de um LSP unidirecional, devido à dificuldade de seleção de rotas. Em LSPs bidirecionais, não se usa *upstream* e *downstream*, pois o fluxo de dados ocorre em ambas as direções. Utilizam-se, então, os conceitos de nó "iniciador" (*initiator*) e de nó "finalizador" (*terminator*). Para o estabelecimento das LSPs bidirecionais, dois rótulos são alocados. Isto é realizado pela presença de um novo objeto (RSVP)/TLV (CR-LDP) denominado *Upstream Label*. O GMPLS estende os protocolos de sinalização, a fim de suportar o estabelecimento de LSPs bidirecionais ao utilizar um conjunto de mensagens de sinalização.

6) Notificação rápida de falhas

A arquitetura GMPLS define algumas extensões para acelerar a notificação de falhas e de outros eventos aos nós responsáveis por restaurar os LSPs com falhas e lidar com os erros. Introduz a capacidade de transmitir informações via conjunto de rótulos aceitáveis (*acceptable label value*), carregada em mensagens de erros específicos nos protocolos de sinalização, contendo uma indicação de valor de rótulo inaceitável (*unacceptable label value*).

O protocolo RSVP permite expedir notificação de falhas e outros eventos para determinados nós. O protocolo CR-LDP não contém este mecanismo de notificação. Para notificar outros nós, a extensão "mensagem de notificação" é utilizada.

7) Rotas explícitas com controle de rótulo explícito (*Explicit Label Control*)

Esta função permite ao LSR de ingresso especificar o rótulo para um, algum ou todos os enlaces de uma rota. Esses rótulos explícitos são especificados pelo LSR de ingresso como parte de uma rota explícita. Em cada LSR ao longo do caminho, qualquer rótulo explícito, especificado em tal rota, tem seu conteúdo analisado e enviado para o próximo nó. O LSR que recebeu o rótulo explícito o utiliza. Caso não seja possível, a configuração do LSP deve ser interrompida.

8) Proteção de enlace

A informação de proteção de enlace é carregada em um novo objeto (RSVP)/TLV (CR-LDP), denominado informação de proteção (*protection information*). Os vários tipos de mecanismos de proteção já foram mencionados no item sobre proteção de enlace dos protocolos de roteamento. O GMPLS propaga a capacidade de proteção de cada enlace em protocolos de roteamento. Desta maneira, algoritmos de determinação de caminhos podem levar em conta esta informação na configuração de um LSP. Informações de proteção também indicam se o LSP é primário ou secundário, em que o secundário é *backup* do primário, não sendo usado até o primário falhar, apesar de poderem ser usados por outros LSPs, até a eventual falha ocorrer no primário.

A Figura 9 mostra o formato da função de “informação de proteção”.

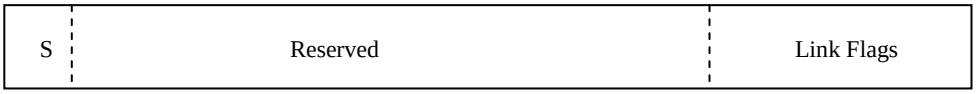


Figura 9 – Formato da Função de Informação de Proteção.

O campo *Secondary* (S) possui tamanho de 1 *bit* e é quando 1 representa um LSP secundário. O campo *Reserved* possui tamanho de 25 *bits.* O campo *Link Flags* possui tamanho de 6 *bits* e indica a proteção para o enlace. Se esse campo for codificado com 0, não possui nenhuma proteção. Os possíveis valores são: (0x20) proteção estendida; (0x10) proteção dedicada 1+1; (0x08) proteção dedicada 1:1; (0x04) proteção distribuída; (0x02) sem proteção e (0x01) extra tráfego.

9) Parâmetros do tráfego específicos por tecnologia

De acordo com a RFC 4328, alguns parâmetros são específicos da tecnologia e transportados adicionalmente no rótulo genérico de requisição. Os parâmetros de tráfego são definidos como se observa na Figura 10:

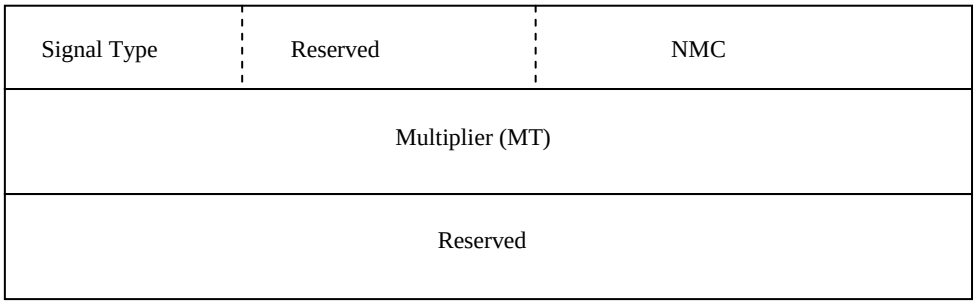


Figura 10 – Formato da Função dos Parâmetros do Tráfego.

O campo NMC apresenta os números de componentes diversificados; o campo NVC representa o número de componentes virtuais, e o campo *Multiplier* (MT). Por sua vez, o campo *Signal Type* (ST) possui o tamanho de 8 *bits* e indica o tipo de sinal que compreende o LSP solicitado. Por fim, o campo *Reserved* (8 *bits* na linha 1 e 32 *bits* na linha 3) é dedicado para uso futuro, devendo ser ajustado para zero quando enviado e ignorado no recebimento.

10) *Status* administrativo de um LSP

A informação de *status* administrativo de LSP é utilizada em dois modos. No primeiro, é indicado o estado de um LSP: se está ativo/inativo, em modo de teste ou se sua própria supressão está em progresso. No segundo, a informação do *status* administrativo indica um pedido para configurar um estado administrativo para um LSP. Tal informação é sempre relatada pelo nó de ingresso, do qual originou o pedido.

2.2. LMP

O protocolo de gerência LMP atua dentro do plano de controle GMPLS. Há poucos trabalhos na literatura da área citando a utilização do protocolo LMP, sendo que os primeiros trabalhos surgiram próximos do ano 2000. Esta utilização foi iniciada por Fredette (FREDETTE et al., 2000) nas redes com plano de controle MPLS, para o aprovisionamento dinâmico de recursos e fornecimento de uma rede de sobrevivência, aplicando técnicas de proteção e restauração.

Outra publicação (LANG et al., 2000) apresentou e especificou o protocolo LMP entre OXCs vizinhos, cuja característica era de aprovisionamento do enlace e de isolação da falha. Benerjee (BENERJEE et al., 2001) utilizou o LMP para localizar falhas em redes transparentes (toda-óptica) e opacas (*optoelectrical*), além de realçar os protocolos de

sinalização RSVP e LDP para a sustentação do GMPLS. Entretanto, só depois da criação da RFC 4204, que padroniza este protocolo, surgiram mais estudos em várias áreas. Esta RFC, também produzida por Lang (LANG, 2005), especifica as MEFs (Máquinas de Estado Finito) das quatro funcionalidades do protocolo LMP: o gerenciamento do canal de controle, a correlação das propriedades dos enlaces a verificação da conectividade física e a localização de falhas dos enlaces, com finalidade de formar uma única TE. Ele descreveu que, para permitir a comunicação entre os nós, garantindo o roteamento, sinalização e o gerenciamento de enlace, deve haver um par de interfaces IP que são mutuamente alcançáveis. Esse par de interfaces é denominado canal de controle. Note que o termo “mutuamente alcançável” não implica que essas duas interfaces são conectadas (diretamente) por uma ligação do IP; uma vez que pode haver uma rede entre o par. Além disso, a interface sobre o qual as mensagens de controle são enviadas/recebidas não pode ser a mesma interface pela qual os dados são transmitidos.

Em GMPLS, os canais de controle entre dois nós vizinhos são separados do meio físico que trafega os enlaces de dados. Quando se permite que os canais de controle entre dois nós sejam logicamente ou fisicamente diferentes e separados dos enlaces de dados, as propriedades de um canal de controle não se correlacionam necessariamente com as propriedades dos enlaces de dados, e vice-versa. Por consequência, uma separação limpa entre o canal de controle e os enlaces de dados deve ser feita. Mecanismos novos devem ser considerados no controle de enlaces de dados (gerência de enlace, aprovisionamento e falha).

O LMP é usado para Engenharia de tráfego (TE) e para verificar as conexões canal de controle. Dentre as tarefas que o LMP realiza, estão os agrupamentos nos enlaces TE dos nós cujas propriedades são correlacionadas, denominada “correlação da propriedade do enlace”. O LMP pode comunicar as propriedades dos enlaces ao módulo IGP, que, por sua vez, pode anunciar aos outros nós da rede (LUO & KUO, 2003).

O LMP é projetado para suportar adicionalmente um ou mais enlaces de dados em um enlace TE. A finalidade do enlace TE é agrupar/mapear informações que são utilizadas para a computação do trajeto e da sinalização GMPLS.

2.2.1. Visão Geral do LMP

O LMP requer, pelo menos, um par de nós ativos entre um canal de controle bidirecional. Cada sentido do canal de controle é identificado pelo campo CC_Id. Uma

“adjacência LMP” é formada entre dois nós quando, pelo menos, um canal bidirecional é estabelecido entre eles. Múltiplos canais de controle podem ser ativos simultaneamente para cada adjacência, entretanto, os parâmetros devem ser negociados individualmente para cada canal de controle.

As mensagens de LMP são transmitidas com confiabilidade, usando mensagens denominadas Message_Ids. As Message_Ids são identificadores carregados dentro da mensagem LMP. O valor da Message_Id é incrementado e zerado quando o valor máximo for alcançado.

O LMP foi especificado pelo IETF e oferece basicamente, quatro funcionalidades que são desempenhadas por meio de vinte mensagens definidas por ele. Dentre suas funcionalidades a RFC 4204 (LANG, 2005) aponta que duas são opcionais e duas são obrigatórias (PASQUINI, 2006):

• verificação de enlace (opcional);
• localização de falha (opcional);
• gerenciamento do canal de controle (obrigatória);
• correlação de propriedades dos enlaces (obrigatória).

Os dois procedimentos principais do LMP são a gerência do canal de controle e a correlação da propriedade do enlace. A gerência do canal de controle é usada para estabelecer e manter os canais de controle entre nós adjacentes. A correlação da propriedade do enlace é utilizada para sincronizar o enlace TE e suas propriedades, além de verificar a configuração desse enlace TE. Esses e outros procedimentos serão detalhados a seguir.

2.2.2. Gerenciamento do Canal de Controle

Este procedimento é realizado para estabelecer e manter canais de controles entre nós adjacentes. Canais de controle existem independentemente de enlaces TE, sendo utilizados para trocar informações provenientes do plano de controle, como, por exemplo: sinalização, roteamento e gerenciamento de enlace. Estas informações dizem respeito ao:

- Ao aprovisionamento de enlaces e informações de gerenciamento de falhas.
- Ao gerenciamento de caminho e de informação de distribuição de rótulos (implementado com protocolos de sinalização RSVP ou CR-LDP).
- À topologia de rede e de informações de estado (usando OSPF-TE e IS-IS-TE).

Há quatro mensagens de LMP usadas para controlar o canal de controle. São as mensagens Config, ConfigAck, ConfigNack e Hello.

Para estabelecer um canal de controle, o endereço IP de destino na extremidade do canal de controle deve ser conhecido. Este conhecimento pode ser configurado manualmente ou descoberto automaticamente. Isto é feito, emitindo a mensagem Config, com endereço de fonte IP no modo *unicast* e o endereço de destino IP no modo *multicast* (224.0.0.1 ou ff02:: 1) (DRAKE et al., 2000).

Os canais de controle existem independentes dos enlaces. Quando esses canais são múltiplos, eles podem ser ativos simultaneamente entre um par de nós. Os parâmetros do canal de controle devem ser negociados em cada canal individualmente, utilizando os pacotes de LMP Hello, a fim de manter a conectividade de LMP, caso outros mecanismos não estejam disponíveis (KOMPELLA, 2005).

Para o LMP, é essencial que um canal de controle esteja sempre disponível para um enlace, e, no caso de uma falha do canal de controle, o canal de controle de substituição (ou o apoio) deve estar disponível, com o objetivo de restabelecer a comunicação com o nó vizinho. Se ainda o canal de controle não puder ser estabelecido, então, tenta-se o canal de controle de *backup.* Podem-se ativar os canais de controle adicionais, seguindo-se as instruções a seguir.

Negociação dos parâmetros

A ativação do canal de controle começa com uma troca de negociação do parâmetro, usando mensagens Config, ConfigAck e ConfigNack. As mensagens são construídas com os parâmetros de LMP, que podem ser negociáveis ou não (identificado pelo *bit* N no cabeçalho do objeto). Os negociáveis podem ser inseridos para que vizinhos LMP concordem com determinados valores. Os não-negociáveis são usados para o anúncio dos valores específicos que não precisam ou não permitem negociação.

Para ativar um canal de controle, conforme o esquema da Figura 11, uma mensagem Config deve ser transmitida ao nó remoto, e, na resposta, uma mensagem de ConfigAck deve ser recebida no nó local. A mensagem Config contém a identificação local do canal de controle (CC_Id), o identificador do nó Node_Id, um Message_Id para a mensagem de confiança, além dos parâmetros. É possível que ambos os nós, locais e remotos, iniciem o procedimento de configuração ao mesmo tempo. Para evitar ambiguidades, o nó com o mais elevado Node_Id ganha a disputa. Se os Node_Ids forem iguais, um nó (ou ambos) serão reconfigurados, mudando o operador Node_Ids.

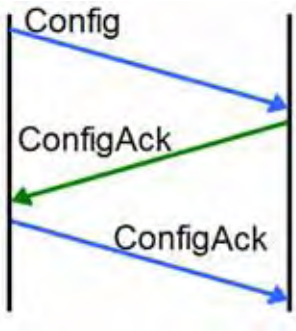


Figura 11 – Ativação do Canal de Controle.

A mensagem de ConfigAck é usada para reconhecer o recebimento da mensagem Config e concordar com todos os parâmetros configurados (negociáveis e não-negociáveis). A mensagem de ConfigNack é empregada para reconhecer o recebimento da mensagem Config, indicando quais parâmetros negociáveis foram (eventualmente) inaceitáveis, propondo valores alternos para a negociação.

Por um lado, se um nó recebe uma mensagem de ConfigNack com substituição aceitável dos valores dos parâmetros negociáveis, o nó deve transmitir a mensagem Config com tais valores. Por outro lado, se um nó recebe uma mensagem de ConfigNack com substituição inaceitável dos valores, o nó pode continuar a retransmitir mensagens dos Config na esperança de que a configuração seja corrigida por um operador, mudando os parâmetros de um ou ambos os nós.

Protocolo Hello e negociação de seus parâmetros

Logo que o canal de controle for ativado entre dois nós, o protocolo Hello será usado para manter a conectividade do canal e detectar falhas rapidamente, enviando mensagens Hello.

Antes de emitir mensagens Hello, os parâmetros HelloInterval e HelloDeadInterval devem ser ajustados, sendo trocados na mensagem Config. O HelloInterval indica a frequência em que as mensagens Hello serão emitidas, medida em milissegundos (ms). Já, o HelloDeadInterval aponta o tempo em que um dispositivo deve esperar antes de declarar um canal de controle inoperante, medido em milissegundos (ms).

O tempo de HelloDeadInterval deve ser, pelo menos, três vezes maior do que o de HelloInterval. Caso o mecanismo Hello não seja usado, os valores do HelloInterval e HelloDeadInterval serão ajustados a zero. Tais valores devem ser selecionados com cuidado, para fornecer rapidamente o tempo de resposta ao canal de controle em caso de falhas, sem causar congestionamento. Os valores sugeridos pela RFC 4204 (LANG, 2005) são: 500ms para HelloInterval e 1500ms para HelloDeadInterval.

O protocolo Hello consiste em uma única mensagem de *unicast* emitida periodicamente ao longo do canal de controle. Cada mensagem contém dois números de sequência: o primeiro será o número de sequência para a mensagem enviada (TxSeqNum), e o segundo será o número de sequência da última mensagem recebida (RxSeqNum). O número de sequência começado por 1 e 0 é indicado para a restauração do nó.

Quando um nó emitiu ou recebeu uma mensagem de ConfigAck, as mensagens Hello podem ser emitidas. Se, entretanto, um nó recebe uma mensagem de ConfigNack ao invés de uma ConfigAck, o nó não deve emitir mensagens.

Mudar para Ativo Rapidamente

Através do TxSeqNum e do RxSeqNum, é possível verificar se o nó remoto está enviando mensagens Hello, ou seja, se este nó está ativo.

O número de seqüência TxSeqNum inicializa sempre com o valor 1, já o RxSeqNum com o valor 0. O número TxSeqNum é usado para indicar que o remetente iniciou ou reiniciou uma conexão. Quando TxSeqNum alcançar (2^{32}) - 1, o número de seqüência seguinte usado é 2, não 0 ou 1, já que estes têm significados especiais.

Sob a operação normal, a diferença entre o RxSeqNum na mensagem que é recebida e o local TxSeqNum gerado deve ser, igual a 1. Se esse valor for diferente significa que houve perdas, reinício ou o nó remoto não está mais ativo, portanto primeiramente o pacote é rejeitado, até que o HelloDeadInterval indique perda de conexão, reiniciando rapidamente o canal de controle.

Canal de Controle Desativado

Para desativar um canal de controle com finalidade administrativa, um *flag* ControlChannelDown está disponível dentro do cabeçalho comum em pacotes LMP. Quando os enlaces de dados estiverem ainda em uso dentro de um par de nós, um canal de controle

pode ser desativado administrativamente se houver outros canais de controle ativos que possam controlar esses enlaces de dados.

Quando um nó receber um pacote de LMP com o flag ControlChannelDown ativo, o nó deverá ajustar este flag em todas as mensagens LMP emitidas sobre o canal. O nó que inicia a desativação do canal de controle pode parar de emitir mensagens após os milissegundos do HelloDeadInterval passarem, ou, então, ao receber um excesso de mensagens do mesmo canal de controle com o *flag* ControlChannelDown ativado.

Estado degradado

Uma consequência, ao se permitir que canais de controle sejam fisicamente separados dos enlaces de dados associados, é que não pode deixar de existir canais de controle ativos disponíveis quando os enlaces de dados ainda estiverem em uso. Para muitas aplicações, é inaceitável desativar um enlace, que está carregando o tráfego do usuário, simplesmente porque o canal de controle não se encontra disponível. Entretanto, o tráfego que está usando o enlace de dados pode não ser garantido no mesmo nível do serviço, deixando, assim, o enlace TE em um estado degradado.

Quando o enlace TE está no estado degradado, o roteamento e a sinalização devem ser notificados de modo que as conexões novas não sejam aceitas.

2.2.3. Correlação das Propriedades do Enlace

A função de correlação das propriedades do enlace é projetada para agregar múltiplos enlaces de dados (portas ou compomentes dos enlaces) em um único enlace TE, e sincronizar as propriedades do enlace TE entre nós vizinhos. Como parte do LMP, um processo de correlação (para determinar inconsistências) de trocas de propriedades de enlaces para enlaces TE é definido, por meio de mensagens LinkSummary, LinkSummaryAck e LinkSummaryNack, mostradas na Figura 12. Suas propriedades podem ser negociáveis ou não, assim como o gerenciamento do canal de controle.

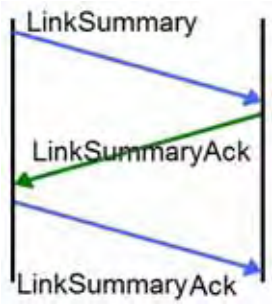


Figura 12 – Processo de Correlação.

Cada enlace de TE possui um identificador Link_Id atribuído a cada nó do enlace. Este identificador deve ser do mesmo tipo (isto é, IPv4, IPv6) em ambos os nós. Se o identificador for diferente, a mensagem LinkSummaryNack deverá ser enviada com o devido erro. Similarmente, em cada enlace de dados, é atribuído um identificador Interface_Id em cada nó. Estes também devem ser do mesmo tipo em ambas as extremidades, caso contrário, outro erro é enviado pela mensagem LinkSummaryNack.

A correlação da propriedade deve ser feita antes da criação de um enlace TE, podendo ser repetida a qualquer momento, desde que o processo de verificação não esteja sendo executado.

A mensagem LinkSummary verifica a consistência do enlace TE e da informação do enlace de dados em ambos os lados. As mensagens são também usadas para:

(1) agregar enlaces de dados múltiplos em um enlace TE;

(2) trocar, correlacionar ou mudar parâmetros do enlace TE;

(3) trocar, correlacionar ou mudar Interface_Ids.

A mensagem inclui o parâmetro TE_LINK seguido por um ou mais parâmetros DATA_LINK. O TE_LINK identifica o enlace TE, Link_Id local e remoto, além de indicar a sustentação para a gerência de falha e para os procedimentos de verificação do enlace para aquele enlace TE. Os DATA_LINK são usados para caracterizar os enlaces de dados que compreendem o TE.

Se a mensagem de LinkSummary for recebida de um nó remoto, que combina com o mapeamento de Interface_Id armazenada localmente, então, dois nós concordam em realizar o procedimento de verificação e de configuração da identificação do enlace de dados. A mensagem de LinkSummaryNack indica quais mapeamentos não estão corretos ou quais propriedades de enlaces não foram aceitas. Caso o mapeamento esteja incorreto, o processo de verificação do enlace é realizado. Já, se a mensagem incluir uma negociação de parâmetros,

os valores aceitáveis devem ser incluídos, para, então, o iniciador da mensagem emitir uma nova LinkSummary com esses valores.

Cunha (CUNHA, 2006) apresentou uma conclusão sobre a opcionalidade da verificação da conectividade do enlace, onde a função da correlação de propriedade de enlace deve ser acionada antes do enlace TE ser estabelecido e colocado em funcionamento a qualquer momento em que o enlace TE esteja ativo. Porém, tanto o Link_ID quanto a Interface_ID, ambos compõe o enlace TE e devem ser correlacionados, mas estes itens somente são conhecidos através de uma configuração manual ou dinamicamente, através da verificação da conectividade do enlace que deve, portanto, ser obrigatoriamente realizada antes da correlação de enlace.

Vale destacar que a RFC 4204, a qual Lang considera a inicialização do enlace TE obrigatória, será implementada apenas quando a verificação do enlace for adicionada.

2.2.4. Verificação da Conectividade do Enlace.

Como um procedimento opcional, pode-se verificar a conectividade física do enlace de dados e descobrir dinamicamente as associações *de* Interface_Id (local e remoto), além dos enlaces TE.

Esse procedimento é realizado se existir um ou mais canais de controle ativos entre dois nós. Acontece primariamente nos enlaces de dados pertencentes ao enlaces TE, e, subsequentemente, nos demais enlaces de dados. O *flag* (TE_LINK) de "verificação de conectividade de enlace", na mensagem LinkSummary, é usado para indicar o procedimento.

O processo de verificação de enlace envolve um determinado número de passos:

1. A troca de mensagens BeginVerify/ BeginVerifyAck/ BeginVerifyNack entre os nós para iniciar a verificação do enlace. A mensagem BeginVerify enviada pelo inicializador (Link_Id) do enlace TE, cuja fibra deseja-se testar, após receber o BeginVerifyAck, pode iniciar o envio das mensagens Test.
2. O envio de mensagens Test nos enlaces, para testar a conectividade física, um após o outro. A mensagem é transmitida no enlace de dados a ser verificado, não pelo canal de controle.
3. A troca de mensagens TestStatusFail, que indica quando a mensagem Test não for detectada no nó remoto durante certo período de observação (pelo VerifyDeadInterval), mostra que a verificação da conectividade física do enlace falhou, marcando o enlace de dados como failed. As mensagens

TestStatusSuccess, opostas à falha, indicam que foi detectada no nó remoto, marcando como up o enlace de dados. Por fim, a mensagem TestStatusAck é enviada reportando o resultado do teste (*failed* ou *up*).

4. A troca de mensagens EndVerify/ EndVerifyAck decretando o fim do processo de verificação.

Essas trocas de mensagens são demonstradas resumidamente na Figura 13.

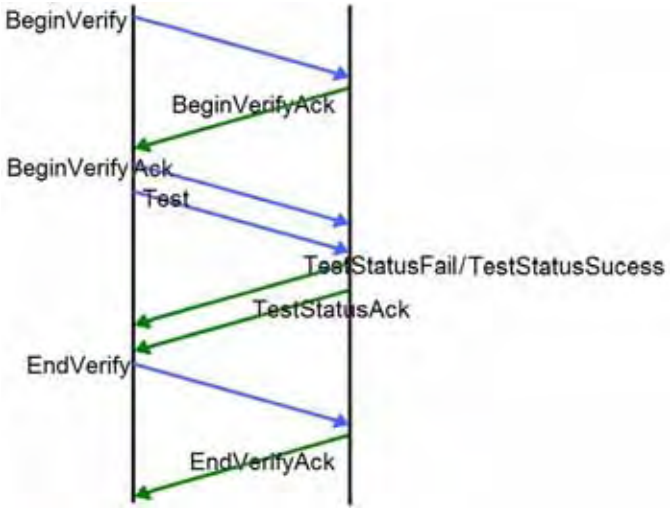


Figura 13 – Verificação da Conectividade.

Os nós locais e remotos devem manter a lista completa de mapeamento de Interface_Id para as finalidades da correlação.

2.2.5. Gerenciamento de falhas

Nesta seção, outro procedimento opcional de LMP é descrito. Ele é realizado para controlar falhas pela notificação rápida do *status* de um ou de mais canais dos dados em um enlace TE, tanto com LSPs unidirecionais ou bidirecionais. Seu espaço é negociado como a parte da troca de mensagem ChannelStatus. A notificação relacionada àquela falha é informada até o nó *upstream.*

Detecção de falha

A detecção de falha deve ocorrer na camada mais próxima à falha. No caso de redes ópticas, esta será a camada física (óptica). Uma medida da detecção de falha na camada física detecta uma perda da luz (LOL). Outras técnicas para monitorar sinais ópticos ainda estão sendo desenvolvidas. É importante destacar que o mecanismo usado para a notificação da falha em LMP é independente do mecanismo usado para detectar a falha, mas se confia simplesmente no fato de que uma falha é detectada.

Procedimento de localização da falha

Se um enlace de dados falha entre dois nós, o sistema de monitoramento de potência em todos os nós *downstream* pode detectar a LOL (em caso de fibra óptica), indicando uma falha. Para evitar múltiplos alarmes originados de uma mesma falha, o protocolo LMP provê uma notificação via mensagem ChannelStatus. Esta aponta que há falhas em um único canal, em múltiplos canais de dados ou num enlace inteiro (DRAKE et al., 2000).

O processo de correlação de falha é feito localmente em cada nó. Ao receber uma notificação de falha, para localizar uma falha em um enlace particular entre nós adjacentes, um nó *downstream* (que detecta a falha no enlace de dados) emitirá uma mensagem ChannelStatus ao seu vizinho *upstream*, indicando que uma falha foi detectada (empacotando junto a notificação de todos os enlaces de dados falhos). Um nó *upstream*, receptor desta mensagem, deve emitir a mensagem ChannelStatusAck ao nó *downstream*. O nó *upstream* deve correlacionar a falha e verificar se ela foi detectada também localmente para o LSP correspondente. Uma vez que a falha é correlacionada, o nó *upstream* emite uma mensagem ChannelStatus ao nó *downstream*, apontando se o canal está em falha ou se está aprovado. Se uma mensagem ChannelStatus não for recebida pelo nó *downstream*, ele transmite uma mensagem ChannelStatusRequest para o canal em questão. No momento em que a falha for localizada, os protocolos de sinalização poderão ser usados para iniciar a proteção e os procedimentos da restauração do caminho (DRAKE et al., 2000), (KOMPELLA, 2005). O trajeto de detecção da falha está ilustrado na Figura 14.

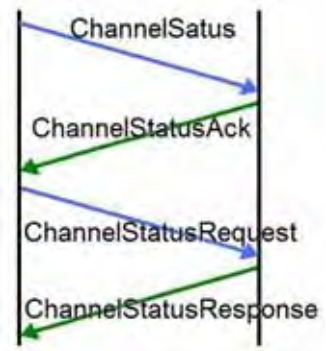


Figura 14 – Ciclo da Detecção de Falhas.

Se todos os enlaces de dados de um enlace TE falharem, o *upstream* poderá ser notificado da falha sem especificar a cada enlace de dados do enlace TE que falhou.

Indicação de ativação e de desativação do canal

A mensagem ChannelStatus também pode ser usada para avisar um vizinho de LMP que um enlace de dados deve ser monitorado ativamente. Isto é chamado de “redes com nós transparentes”, em que o *status* dos enlaces de dados pode indicar a ativação do canal. É particularmente útil quando se necessita despertar o canal de controlem usando mensagens.

A mensagem de ChannelStatusAck deve ser transmitida depois do recebimento da mensagem ChannelStatus. Quando a mensagem ChannelStatus é recebida, os enlaces de dados correspondentes devem ser colocados no modo ativo. Se isso não ocorrer, uma falha é detectada (vide descrição na seção 4.5.1.).

A mensagem ChannelStatus também pode ser utilizada para notificar um vizinho de LMP que um enlace de dados já não necessita ser monitorado ativamente. Quando uma mensagem ChannelStatus for recebida com “desativação do canal”, os enlaces de dados correspondentes saem do estado ativo.

2.3. Trabalhos Recentes envolvendo LMP

Nos últimos anos, diversos trabalhos envolvendo LMP dentro da arquitetura GMPLS vêm sendo produzidos. Os trabalhos que serão apresentados a seguir abordam avaliações do protocolo, novas propostas e mix de tecnologias. Em seguida, apresenta-se o LMP sendo utilizado na tecnologia *Bluetooth*, principalmente para estabelecer e manter as transferências de dados, voz e vídeos durante a conexão.

2.3.1. Trabalhos referentes ao LMP na arquitetura GMPLS

As atualizações referentes à RFC 4204 foram propostas logo no ano de 2005. A primeira, RFC 4209 (FREDETTE & LANG, 2005), apresentou extensões para o protocolo LMP a fim de permitir ser usado não somente em um par de nós e sim em sistemas de linhas ópticas adjacentes. As extensões propostas modificam parâmetros e mensagens sobre as quatro funcionalidades do LMP. Já a RFC 4327 (DUBUC et al.,2006) desenvolveu um módulo MIB que pode ser usado para gerenciar implementações LMP. Este módulo MIB abrange tanto a configuração e desempenho de aspectos de segurança do LMP. Por fim, a última RFC 4394 (FEDYK et al., 2006) apresentou uma mapeamento entre o LMP e ASON (G.805/G.8080), sendo este um passo importante para compreensão das duas arquiteturas e uma cooperação entre as áreas. Foi construída uma tabela de equivalência de parâmetros de uma tecnologia a outra. As tecnlogias foram comparadas e concluiu-se que LMP é uma realização válida de um processo de descoberta sob plano de controle baseado no modelo G.8080.

Desde a criação das RFCs muitas pesquisas surgiram, como a de Perelló (PERELLÓ et al., 2006), que apresentou um mecanismo *bootstrap* para que o canal de controle funcionasse sem as informações de topologia externa LMP. Algo diferenciado, já que o protocolo de gerência de enlace requer a configuração manual dos endereços IPs do canal de controle remoto, além de apresentar extensões para LMP Config e mensagens ConfigAck, que permitem a inicialização, relacionadas com controle de *backup.*

Em outro trabalho, Perelló (PERELLÓ et al., 2007) propôs avaliar as novas extensões para o padrão LMP na descoberta de planos de transporte em redes transparentes (toda-óptica) num projeto chamado CARISMA, o qual usa o LMP para, automaticamente, detectar e recolher informações do estado dos nós vizinhos, além de criar a base de dados. Esta questão assume um papel especial porque implica em redução de despesas em uma ampla área de redes, na qual grandes distâncias entre os elementos da rede implicam em vários problemas logísticos, e qualquer intervenção manual seria muito complexa. Ele também avaliou o impacto das falhas de enlace no plano de controle em topologia de rede “tipo anel” de diferentes cenários e concluiu que a qualidade de serviço estava sendo prejudicada devido ao *overhead* de controle gerado. Sua proposta foi compensar esse valor no HelloInterval do LMP.

Tsuritani e Otani (TSURITANI & OTANI, 2007) apresentou os resultados dos testes que demonstram o funcionamento entre equipamentos WXC (*Wavelength Cross-Connect*) e

PXC (*Photonic Cross-Connect*) para a comutação de redes ópticas confiáveis. Os protocolos GMPLS, principalmente, o LMP, foram utilizados para controlar e fornecer diversas funções aos PXCs, tais como: configuração de conexão/desconexão, trocas de informações de topologia e isolamento de falhas. O trabalho provou o aumento da confiabilidade e da capacidade de gerenciamento, utilizando tais equipamentos.

Apresentando a funcionalidade de proteção do enlace, Velasco (VELASCO, 2007) propôs e avaliou uma solução completa para construir redes ópticas baseadas em anel com capacidade da proteção dedicada chamada *OMS-DPRing*. A solução consistiu em um mecanismo – baseado em extensões do LMP – para coordenar as ações da proteção e um novo modelo de Multiplexador de Inserção/Remoção Óptico Reconfigurável (ROADM) para suportar a proteção *OMS-DPRing*.

Trabalhando também com proteção, uma arquitetura GMPLS – baseada em um mecanismo de recuperação sistemática – foi proposta por Shan-Guo (SHAN-GUO et al., 2008), utilizando o protocolo de gerenciamento de enlace. Nela, o LMP foi usado para gerenciar a engenharia de tráfego e os enlaces, verificando a acessibilidade do canal de controle. Como esse protocolo pode suportar uma comunicação bidirecional tornou-se mais eficaz se notificar as falhas bidirecionalmente. Extensões foram aplicadas ao protocolo de enlace para suportar o processo de notificação bidirecional.

Shimizu (SHIMIZU et al., 2008) propôs o uso de técnicas *plug and play* para a configuração do GMPLS. Este plano de controle possui muitos parâmetros que causam um grande *stress* nas redes atuais. Suas técnicas consistiram em uma configuração automática de ajuste dos parâmetros, utilizando extensões no LMP e um protocolo de funcionamento dinâmico. As técnicas *plug and play* minimizaram os custos dessas redes.

O trabalho de Huang (HUANG et al., 2008) utilizou o LMP com a função de detectar falhas de um modo mais eficaz. As principais contribuições envolveram um mecanismo de recuperação de falhas com uma maior utilização dos recursos no roteamento otimizado, redução da latência dessa recuperação, estabilidade satisfatória e flexibilidade. Além disso, incluiu o grupo a risco partilhado do enlace (SRLG).

Outro artigo de Shimizu (SHIMIZU et al., 2008), avaliou redes com os protocolos OSPF/ RSVP e os comparou utilizando OSPF-TE, RSVP-TE e o LMP. Os pontos avaliados foram o controle redundante de múltiplos canais, o mecanismo *keep-alive* (manter-vivo) e a virtualização do plano de controle. Segundo o estudo, o método proposto que utiliza TE e LMP é mais rápido do que o método que usa OSPF e mecanismos Hello, já que a mudança de SPFs permite atrasos. Houve também a proposta de redução do volume de tráfego de até 10%

em relação ao outro método. O método proposto não pode diminuir as despesas de controle de rede, apenas os custos de operação.

Perelló (PERELLÓ et al., 2009), adaptou o protocolo LMP para atenuar sobrecargas causadas por atrasos de propagação em redes híbridas. Para propagar as informações dos estados dos nós na rede, não utilizou o protocolo OSPF, devido à complexidade e lentidão. Então acrescentou ao LMP um pacote de controle de congestionamento que propaga apenas informações dos estados dos vizinhos, ao invés de tentar obter uma visão de toda a rede. A esse novo protocolo deu o nome de LMP- Protocolo de Roteamento Preventivo (PDR). Com o HelloInterval do LMP, que reinicia a casa 150ms, garantiu que atualizações a cada reinício. Os testes realizados provaram que as informações passadas pelo LMP- PDR foram suficientes para atualizar o estado da rede, além de introduzir uma baixa na sobrecarga.

No LMP, também foi acrescido a funcionalidade de monitoramento do desempenho do enlace óptico (LEE et al., 2009). Baseada em uma técnica que mede com precisão vários parâmetros utilizados no cálculo do caminho do enlace bem sucedido, um módulo de controle – capaz de monitorar os parâmetros de múltiplos enlaces – foi implementado e o LMP estendido para acompanhar o monitoramento. A técnica proposta mostrou-se útil para a evolução das redes ópticas GMPLS controladas integralmente ao fornecer as ferramentas facilitadoras do gerenciamento de enlaces.

Outra pesquisa que mediu parâmetros foi a de Seno (SENO et al.,2009), onde o LMP foi implementado para o controle de um dispositivo de dispersão cromática de medição da fibra óptica. Para a medição automática de dispersão cromática, um dispositivo de medição foi colocado em cada enlace de um nó ou compartilhado por todos os enlaces de um nó via interruptor óptico. Dentre os parâmetros requisitados, estão: a identificação inequívoca de um meio físico a ser medido; a sincronização do sinal de medição entre o remetente e o receptor; a medição de recuperação de erros de protocolo; a troca de resultados de medição; a alta compatibilidade com nós GMPLS existentes. Os resultados demonstraram com sucesso o LMP no controle de medição de dispersão cromática de mais de 80 quilômetros de fibra de monomodo.

Alguns projetos mais atuais mostram a união das soluções tecnológicas para as redes ópticas, como, por exemplo: a tecnologia OBS[1] e a GMPLS. A OBS combina as melhores características da comutação dos circuitos e da comutação dos pacotes. A arquitetura GMPLS

[1] OBS – Comutação de Rajadas Ópticas. Dados são enviados separados do controle sem o uso de fibras de retardo. Uma rajada de dados é enviada após um tempo de ajuste sem que haja qualquer confirmação de que os recursos necessários foram alocados corretamente.

fornece uma melhor adaptação das camadas IP e do sistema óptico. Portanto, a OBS apresenta uma alternativa ao paradigma da comutação na camada ótica, enquanto o GMPLS facilita o controle óptico e a manipulação da rede. Unir essas funcionalidades do GMPLS no plano de controle convencional OBS aumentou a escalabilidade e a sobrevivência das redes (JULIÁN, 2009).

No artigo de Zhang e Bao (ZHANG & BAO, 2010), discutiu-se o GMPLS e sua aplicação para o núcleo de rede óptica, combinando controle e gerenciamento da rede IP-sobre-Óptica em um conjunto comum de protocolos de controle. Nele o LMP atualiza o banco de dados EMS, assim que for executada a auto-descoberta da rede, a fim de garantir a consistência e a sincronização de informações. A expectativa foi atingida para a rede de fibra óptica, que ofereceu o restabelecimento rápido, competitivo na velocidade com anéis SONET, aliviando os desafios vivenciados com os serviços atuais e também abre as portas para novos serviços sofisticados, tais como: *Gigabit Ethernet*, armazenamento de dados, vídeo *streaming* e redes privadas virtuais (VPNs).

Hayashi e Shiomoto (HAYASHI & SHIOMOTO, 2010) utilizaram a tecnologia óptica *plug and play* (PnP) para automatizar as informações necessárias relacionadas ao plano de controle e o plano de dados, tais como: atribuição automática de endereços de interface, informações de parâmetros trocadas entre nós vizinhos, a construção de um banco de dados, e assim por diante. Para que isso fosse possível, o LMP foi extendido, adiantando o procedimento de verificação do enlace ao iniciar o canal de controle e construindo uma base de dados com as informações dos nós.

2.3.2. Trabalhos referentes ao LMP na tecnologia *Bluetooth*

O protocolo LMP fornece serviços para o plano de controle GMPLS, porém, fora dessa arquitetura, o LMP é utilizado em plataforma *Bluetooth*, como mostrado no trabalho de Loureiro (LOUREIRO et al., 2003) apresentou redes de sensores sem fio, com um grande número de nós distribuídos, com restrições de energia, auto-configuração e adaptação, devido às falhas e perdas de nós. Nessas redes, cada nó foi equipado com uma variedade de sensores, tais como: acústico, sísmico, infravermelho, vídeo-câmera, calor, temperatura e pressão. Os protocolos de acesso ao meio (Controle de Acesso ao Meio – MAC) foram baseados na comunicação sem fio (*Bluetooth*). Dentro de tal tecnologia, o LMP gerenciou o estabelecimento e o controle de enlaces, bem como a gerência de consumo de energia, o estado do dispositivo na *piconet* e o controle de autenticação e criptografia.

A descrição do projeto ATLANTIS, proposta por Ye (YE, 2005) mostrou o desenvolvimento de uma estrutura centralizada que permitiu a aproximação por meio do sensor de *Bluetooth*. O LMP foi responsável por ajustar dinamicamente os seus níveis de potência, utilizando uma escala de potência denominada *Golden Received Power Range* (GRPR), específica para cada dispositivo. Os resultados comprovaram que o LMP conseguiu atingir uma ótima recepção de dados sem desperdiçar energia.

Espinosa (ESPINOSA et al., 2009) implementou um dispositivo de leitura automática de medidores de consumo de água com tecnologia *Bluetooth*. O protocolo LMP, juntamente com protocolos superiores, permitiu um nível de segurança adicional previsto pelo *Bluetooth* e limitou os processos de transições.

Ting (TING et al., 2010), utilizou a tecnologia de comunicação *Bluetooth*, que baseia-se no dispositivo portátil com telefonia inteligente utilizado em monitoramento de hospitais. O LMP é responsável pelo envio da sinalização para inicializar uma conexão direta ou desligar esta conexão. Um dispositivo pode conectar a 255 filiados. Usando a tecnologia sem fios Bluetooth, foi obtida uma gama de comunicações de voz e dados de alta qualidade e com segurança, além das vantagens de flexibilidade e baixo custo.

Conforme apresentado, houve um grande aumento nos estudos sobre o protocolo LMP. No entanto, ainda não foi disponibilizada uma versão *open-source* desse protocolo que possa ser integrada aos diversos de tecnologias.

3. DESENVOLVIMENTO

Neste capítulo são apresentados detalhes do funcionamento do pacote DRAGON e da implementação do protocolo LMP.

Inicialmente apresenta-se o projeto DRAGON, uma arquitetura GMPLS que permite promover uma infra-estrutura às aplicações de redes para que estas possam adquirir recursos de forma dinâmica e determinística, e integrá-los aos diversos dispositivos que formam a rede. São mostrados os diversos domínios autônomos da rede cada um capaz de definir unilateralmente a política interna da gestão de tráfego e arranjos bilaterais com outros domínios externos que devem ser ajustados mutuamente. Em seguida, mostra-se a implementação do protocolo LMP proposto, que tem como objetivo prover o gerenciamento dos enlaces em uma rede com arquitetura GMPLS, verificando a qualidade das conectividades físicas entre nós vizinhos, identificação das propriedades dos nós adjacentes e isolação de falhas simples ou múltiplas no domínio óptico. Para melhor entendimento do protocolo, foram descritas as MEFs (Máquinas de Estados Finitas), mostrando todas as trocas de mensagens e os parâmetros solicitados.

3.1. DRAGON

Como uma iniciativa da NSF (*National Science Fundation*), o DRAGON é uma colaboração entre um conjunto de instituições de pesquisas americanas para promover uma infra-estrutura que permita que as aplicações de redes adquiram de forma dinâmica e determinística recursos de rede para integrar os diversos dispositivos que formam a rede (LEHMAN et al., 2006).

A escolha pelo pacote DRAGON como parte desse estudo se deu pelo fato do pacote, ser o único pacote GMPLS que utiliza os protocolos (RSVP E OSPF) em versão *open-source, além de* já testado em um *backbone* óptico.

O objetivo do DRAGON é compartilhar, através da rede, *clusters* computacionais, dispositivos de visualização, sensores remotos e outros instrumentos com aplicações

específicas que estão em localizações diferentes do mundo e possuem alto custo de aquisição (DRAGON Project Website, 2008).

Outras aplicações que possuem grande interesse nestes tipos de serviços oferecidos pelo DRAGON são aquelas que necessitam de resposta imediata e engenharia de tráfego para um melhor desempenho da rede IP. Serviços de rede "determinística", ou seja, que garantam um determinado nível de serviço, possuem parâmetros nivelados que incluem: um mínimo da largura de faixa, habilidade de especificar a latência, o *jitter*, a perda do pacote, e outros parâmetros. Estes serviços precisam ser provisionados em uma base do inter-domínio, através das tecnologias das redes heterogêneas, incluindo a autenticação, autorização, e programação (*Authentication, Authorization, Accounting* - AAA).

Para atingir a rede dinâmica e determinística, o DRAGON desenvolveu um plano de controle baseado no GMPLS que incluí técnicas avançadas de roteamento inter-domínio de serviço. Esse plano de controle oferece:

- Gerenciamento da Rede como uma única entidade;
- Localização e geração de relatórios sobre dispositivos e usuários na rede;
- Gerenciamento de *firmwares* e configurações de múltiplos dispositivos simultaneamente;
- Customização de relatórios gerenciais das mudanças na rede;
- Propriedades para Autenticação, Autorização e Programação e agendamento.

3.1.1. Arquitetura DRAGON

A arquitetura DRAGON consiste em diversos domínios autônomos da rede, cada um capaz de definir uma política interna da gestão de tráfego, e arranjos com outros domínios externos que devem ser ajustados mutuamente. A arquitetura DRAGON é baseada em *switches* e nós de transmissão que suportam a hierarquia de rótulo de GMPLS (DRAGON *Control Plane Overview*, 2008).

A hierarquia GMPLS compreende os rótulos do pacote (por exemplo, MPLS), os rótulos de TDM (por exemplo SONET), os rótulos do comprimento de onda, e os rótulos da fibra. Esses rótulos são recebidos pelos LSRs, que por sua vez, não comutam cada tipo de rótulo, más sim, um subconjunto baseado nas capacidades do equipamento e da rede.

Na arquitetura do DRAGON, em cada LSR funciona um protocolo do roteamento intra-domínio GMPLS (por exemplo, GMPLS-OSPF) e um protocolo de sinalização do trajeto de GMPLS (por exemplo, GMPLS-RSVP). Essa combinação de roteadores associados a sinalização, permite o controle dinâmico e a instanciação dos LSPs, que fornecem conexões fim-a-fim para o tráfego multi-serviços.

Para o estabelecimento dos LSPs intra-domínio baseados em *lambda* é necessário adicionar extensões nos protocolos IGP e nos protocolos de sinalização. Entretanto, a instanciação de LSPs inter-domínio dinamicamente em redes heterogêneas ainda é problemática.

O módulo de roteamento é baseado em uma implementação do software ZEBRA. O módulo de sinalização é baseado em modificações feitas ao protocolo KOM-RSVP (*Multimedia Communications Lab - Resource Reservation Protocol*), que sinaliza se a rota escolhida pode ser uma rota "frouxa" ou estrita.

Uma descrição mais detalhada dos componentes do DRAGON pode ser vista a seguir:

NARB (*Network Aware Resource Broker*)

NARB é apresentado na Figura 15, como uma entidade que representa o Sistema Autônomo local (AS) ou domínio. O NARB serve como uma máquina de computação de caminho, da qual sistemas das extremidades ou outros dispositivos possam examinar e descobrir sobre disponibilidade de tráfego, criando caminhos entre fonte especificada e o destino. Um sub-componente no NARB é chamado de Elemento de Computação de Recurso (RCE), que executa as tarefas de computação de caminho. O NARB também é responsável pelo roteamento inter-domínio.

ASTB (*Application Specific Topology Builder*)

A arquitetura DRAGON inclui a noção de estabelecer "Aplicação Específica das Topologias" (ASTB), conforme mostra a Figura 15. ASTB é utilizado por aplicações específicas para pedir serviços de rede. As aplicações podem pedir um caminho dinâmico (LSP) através de uma rede DRAGON. Onde um grupo de mensagens simples permite que uma aplicação peça um trajeto de rede dedicada entre uma fonte e um destino, com uma LSP individual. (DRAGON, 2006).

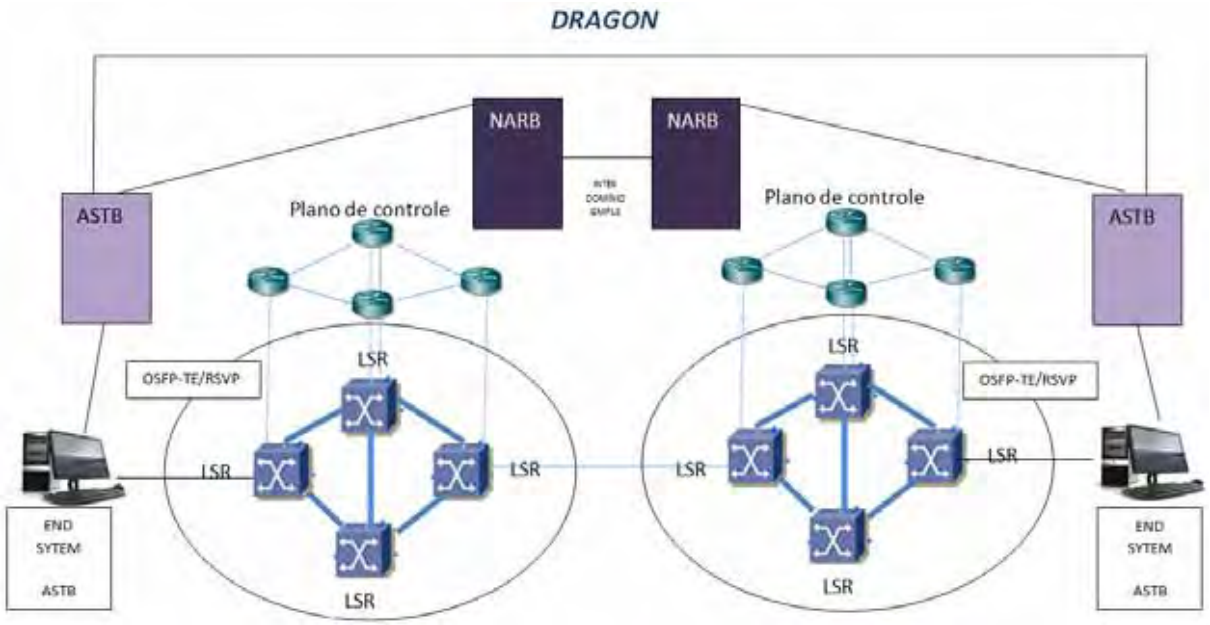


Figura 15 – Arquitetura DRAGON.

Além desses componentes, o DRAGON ainda possui outras duas características, como:

VLSR (*Virtual LSR*)

O VLSR permite a dispositivos que não são equipados com o GMPLS participarem dos processos de roteamento e sinalização em redes de aprovisionamento fim-a-fim GMPLS. O VLSR traduz os níveis de protocolos GMPLS em dispositivos com protocolos específicos, permitindo reconfiguração dinâmica de dispositivos não GMPLS. O VLSR possui os protocolos OSPF-TE e RSVP-TE, que atuam na busca de dispositivos não GMPLS, incluindo esses dispositivos nas instanciações de caminho de fim-a-fim.

CSA (*Client System Agent*)

O CSA é um software que participa dos protocolos de GMPLS para permitir o aprovisionamento da demanda fim-a-fim de um sistema de cliente para outro sistema de cliente. Um "cliente" é qualquer sistema que está pedindo serviço da rede, tal como um *host*, um *cluster* computacional, um roteador, um telescópio e vários outros dispositivos de redes. O

CSA executa tipicamente em modo ponto-a-ponto, mas pode agir pelo meio do protocolo UNI (*User Network Interface*), ou um modo de serviço de rede.

3.1.2. VNE

A Rede Experimental Virtual (VNE) *open-source*, criadas por Ham e Verhoog (HAM & VERGOOG, 2004) foi modificada para executar o código DRAGON no ambiente UML. Este fornece a maneira de criar e executar máquinas virtuais em Linux. As instancias UML podem ser conectadas em redes virtuais *ethernet* o qual são acessíveis por interfaces através do UML configuradas com *ifconfig*.

A Figura 16 mostra o funcionamento do VNE do DRAGON iniciando as janelas NARB, VLRS, *Switch* e estações de computador (es1 e es2) em emuladores de terminais *xterms*.

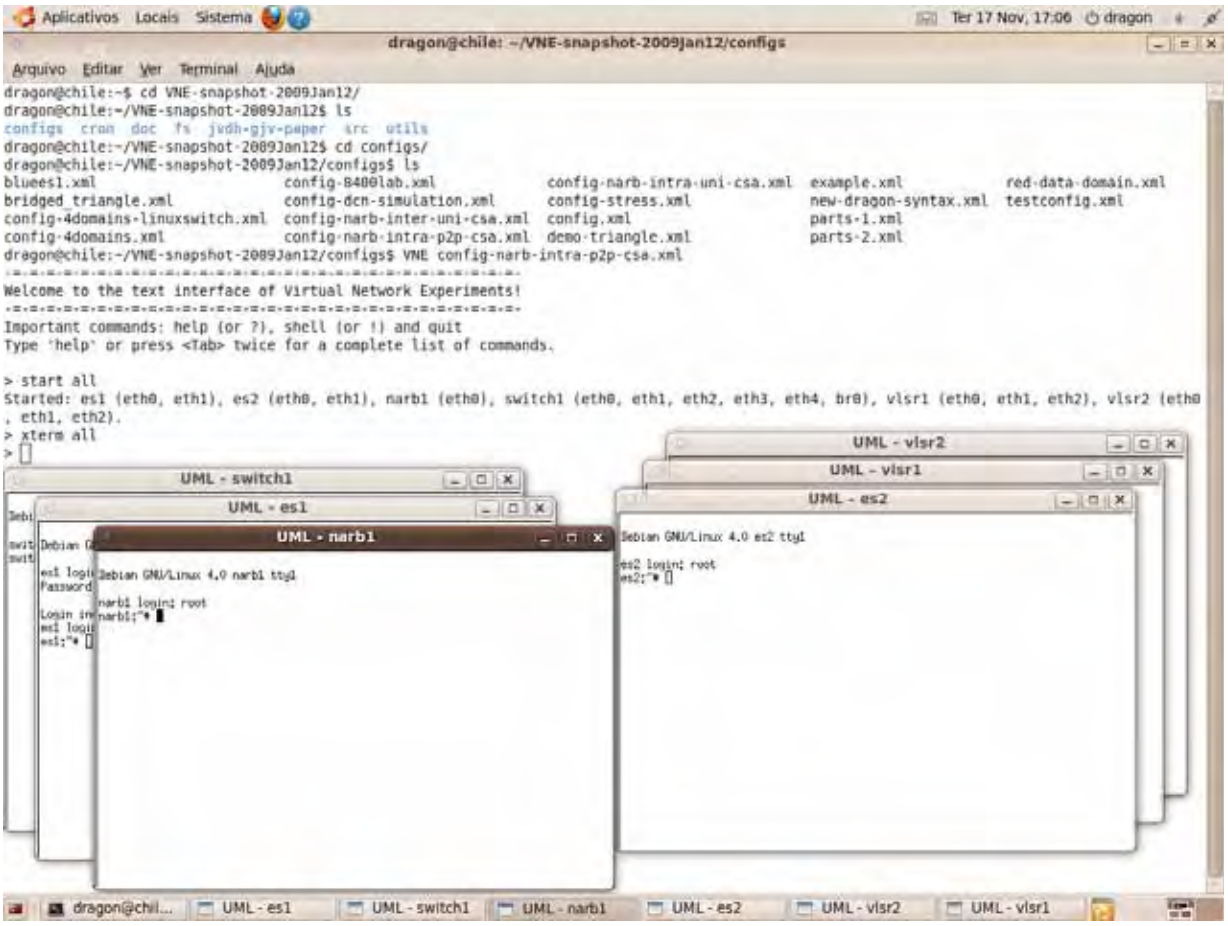


Figura 16 – VNE do DRAGON.

Cada exemplo de UML pode ter um número arbitrário de interfaces virtuais *Ethernet.* O DRAGON utiliza o *switch* virtual UML para conectar através de suas interfaces de redes as máquinas virtuais. Este *switch* tem exatamente a função de um *switch* real da camada 2 e pode ser opcionalmente configurado para atuar como um *hub* passivo.

3.2. Implementação do Protocolo

O pacote *open-source* DRAGON é baseado na linguagem C. Portanto essa também foi a linguagem utilizada no protocolo LMP, principalmente por possuir uma grande quantidade de bibliotecas facilitando a programação, além de um compilador livre, o GCC.

Dentre as bibliotecas e opções da linguagem C, a biblioteca socket foi muito utilizada na programação do DRAGON e também do LMP. O protocolo LMP utiliza o protocolo de transporte UDP, portanto a troca de *datagramas* é realizada através das mensagens *sendto* e *recvfrom*. Dentro do LMP, após estabelecida a comunicação, a função de gerenciamento do canal e correlação de enlace de dados em um único enlace TE (com engenharia de tráfego), são então inicializadas.

A Figura 17 a seguir mostra um protótipo da implementação do protocolo LMP.

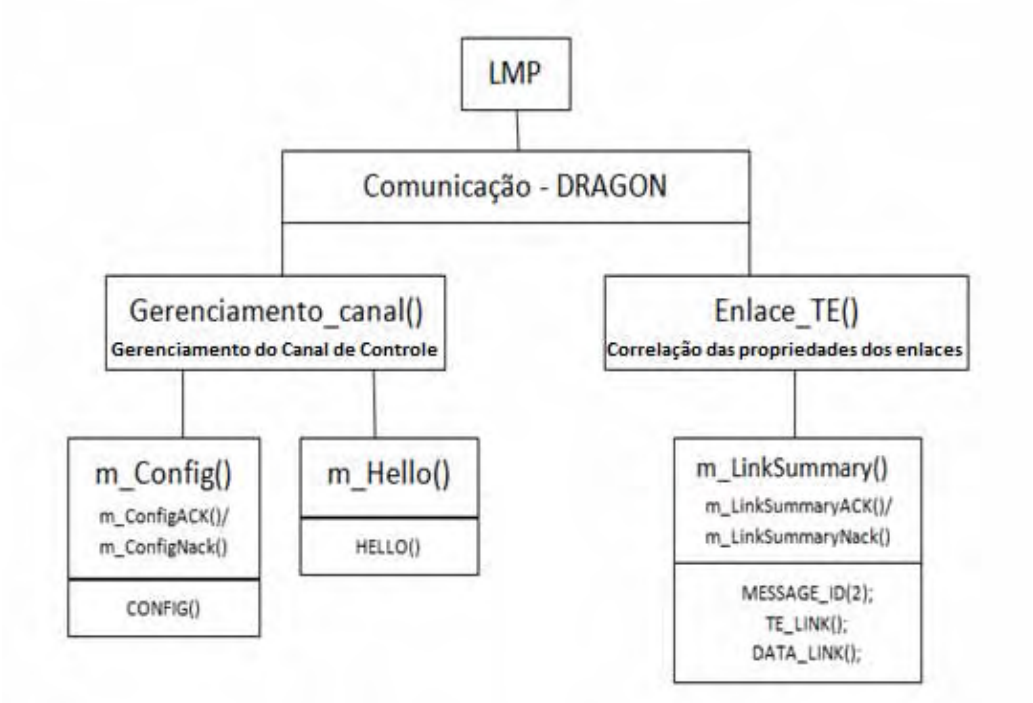


Figura 17 – Protótipo LMP.

3.3. Máquina de Estado Finito (MEF)

A Máquina de Estado Finito (MEF) é a modelagem de um comportamento, composta por estados, transições e ações, auxiliando no desenvolvimento dos algoritmos necessários para o desenvolvimento do LMP.

Este trabalho foi baseado em duas MEFs da RFC 4202 de Lang (LANG, 2005). Estas duas MEFs representam as funções obrigatórias do LMP (gerenciamento do canal de controle e correlação das propriedades dos enlaces). Somente estas duas funções foram implementadas no pacote *open-source* DRAGON.

3.3.1. MEF do Canal de Controle

A MEF do canal de controle define os estados e as lógicas da operação de um canal de controle do LMP.

Estado do Canal de Controle

A função gerenciamento_canal() mostra que um canal de controle pode estar em um dos estados descritos abaixo, conforme a Figura 18. Cada estado corresponde a uma determinada condição do canal de controle associada geralmente com um tipo específico de mensagem LMP que é transmitida periodicamente à extremidade.

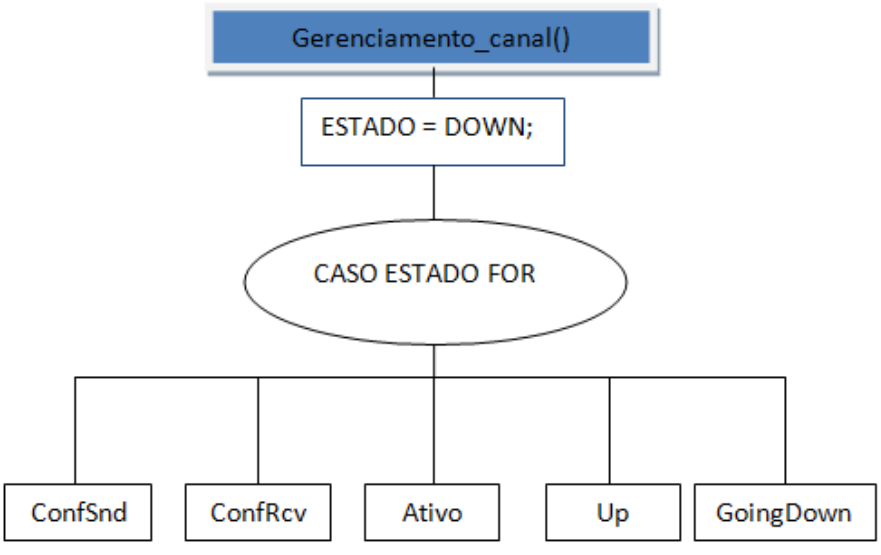


Figura 18 – Fluxograma Gerenciamento do Canal.

Down: Este é o estado inicial do canal de controle. Neste estado, nenhuma tentativa está sendo feita para levar o canal de controle para *up* e nenhuma mensagem de LMP é emitida. Os parâmetros do canal devem ser ajustados aos valores iniciais.

ConfSnd: O canal do controle está na negociação dos parâmetros. Neste estado, o nó emite periodicamente a mensagem dos Config, e espera o outro lado responder com uma mensagem de ConfigAck ou de ConfigNack. O MEF não transita para o estado ativo até que o lado remoto reconheça positivamente os parâmetros.

ConfRcv: O canal do controle está na negociação dos parâmetros. Neste estado, o nó espera aceitar os parâmetros da configuração do lado remoto. Uma vez tais parâmetros são recebidos e reconhecidos, o MEF transita ao estado ativo.

Ativo: Neste estado o nó emite periodicamente uma mensagem Hello e fica esperando receber uma mensagem válida de Hello. Uma vez que a mensagem válida Hello é recebida, ele pode transitar para o estado *up*.

Up: O CC está em um estado operacional. O nó recebe as mensagens válidas Hello e emite mensagens Hello.

GoingDown: Um CC pode entrar neste estado por causa de uma ação administrativa. Quando um CC estiver neste estado, o nó ajusta o *bit* ControlChannelDown em todas as mensagens que emite.

Eventos no Canal de Controle

Dentro da função gerenciamento_canal(), a operação do canal de controle de LMP é descrita nos termos da MEF em estados e eventos. Eventos são gerados pelos protocolos subjacentes e módulos de software, assim como pelas rotinas de processamento de pacotes e MEFs associadas ao enlace TE. Cada evento tem seu número e nome simbólico.

A Figura 19 ilustra a operação do MEF do canal do controle em forma de diagrama de transição do estado MEF.

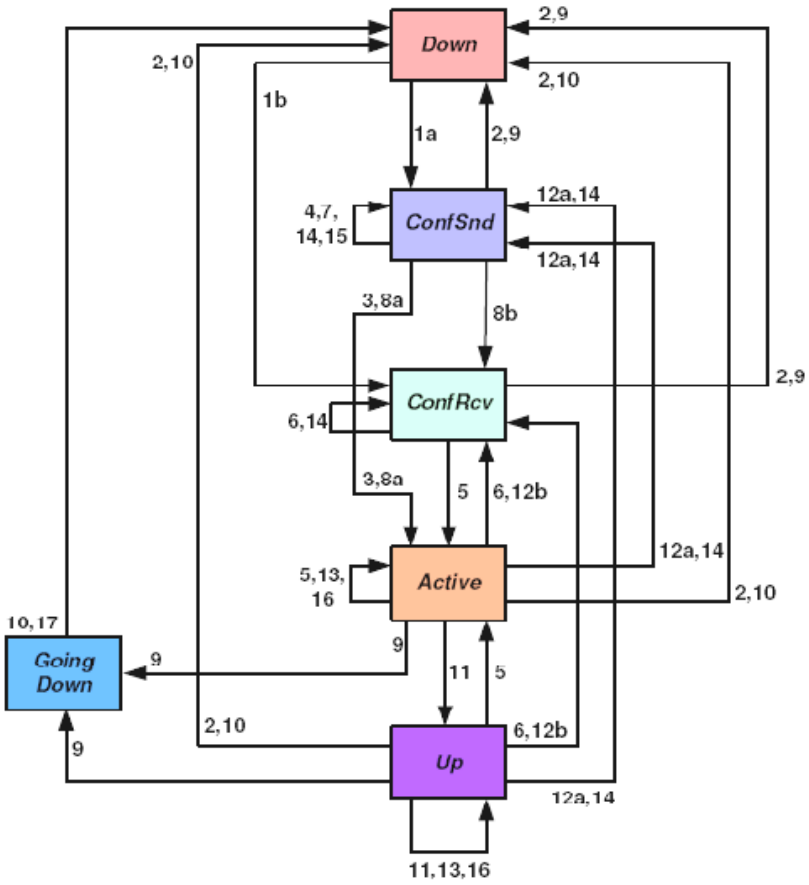


Figura 19 – MEF do Canal de Controle (CUNHA, 2006).

Dentre os eventos de 1 à 8 o canal de controle é estabelecido através do conjunto das mensagens m_Config(), m_ConfigACK e m_ConfigNACK, onde o recebimento da mensagem m_ConfigACK() será respondida com outra mensagem m_ConfigAck() estabelecendo assim o canal de controle. Se a mensagem recebida for m_ConfigNack() o canal deve verificar os parâmetros incompatíveis e analisar o *flag* n para saber se são ou não negociáveis esses parâmetros. Essa verificação é feita pela mensagem negociacao_parametros().

Os parâmetros como, HELLO_INTERVAL e HELLO_DEAD_INTERVAL, são instanciados dentro da mensagem CONFIG() que é chamada pela mensagem m_Config(), conforme o pseudo-código abaixo mostra.

```
programa m_CONFIG
Início
i : inteiro
HelloConfig[2] :  inteiro longo
escreva("CONFIG\n")
    Para i=0 até i>=2 repetir
        HelloConfig[i] ← '0';
    Fim para
    lmp.obj.classe ←CONFIG
    Se c_type for igual a 1 faça
        lmp.obj.c_type  ← c_type
        HelloConfig[0] ← HELLO_INTERVAL
        escreva("%ld ",HelloConfig[0])
        HelloConfig[1] ← HELLO_DEAD_INTERVAL
        escreva("%ld ",HelloConfig[1])
        lmp.obj.n ← 1
        Para i=0 até i<2 repetir
            lmp.obj.objeto[i] ← HelloConfig[i]
            escreva("%ld \n",lmp.obj.objeto[i])
        Fim para
    Chamar função para enviar o socket
    Retornar 1
    Fim Se
    Senão chamar função ERROR_CODE(1, "0x10")
    Retornar 5
    Fim Se
Fim
```

Após o estabelecimento do canal de controle as mensagens config são enviadas periodicamente pela mensagem m_Hello() através dos eventos 11, 12 e 13 da Figura 20 já descrita anteriormente, passando como parâmetros o TxSeqNum e o RcvSeqNum inicializadas pela mensagem HELLO() apresentada abaixo.

```
programa m_Hello
Início
HELLO_G[2] :  inteiro longo
i : inteiro
lmp.obj.classe ← 7; // 7 = HELLO
 libera memória HELLO_G
    Para i=0 até i>=2 repita
        HELLO_G[i] ←'0'
    Fim Para
        Se c_type igual a 1 faça
            lmp.obj.c_type ← c_type
            Se RcvSeqNum diferente de 0 faça
                TxSeqNum ← TxSeqNum + 1
                HELLO_G[0] ← TxSeqNum
                HELLO_G[1] ← RcvSeqNum
            Se TxSeqNum igual a 0 faça
                ESTADO ←Down
            Fim Se
            Se HELLO_G[0] igual a HELLO_G[1] faça
                Copia (lmp.flag, "0")
                TxSeqNum= ←1
                RcvSeqNum←0
                HELLO_G[0] ← TxSeqNum
                HELLO_G[1] ← RcvSeqNum
                ESTADO ←Down
            Fim Se
            Se HELLO_G[0] igual a 4294967295 faça        //((2^32)-1))
                TxSeqNum ←2
                HELLO_G[0] ←TxSeqNum
            Fim Se
        Fim Se
        Senão
            TxSeqNum← 1//Calculo inicial
            RcvSeqNum ←0  // Indica que nenhum Hello foi recebido
            HELLO_G[0] ← TxSeqNum
            HELLO_G[1] ← RcvSeqNum
            lmp.obj.n ← 0
        Fim Se
```

```
    Para i=0 até i<2 repita
        lmp.obj.objeto[i] ←HELLO_G[i]
        escreva ("HELLO%ld ",lmp.obj.objeto[i])
    Fim Para
Chamar função para enviar o socket
escreva("exit hello")
retorna 1
Senão chama função de erro
    retorna 5
Fim
```

1: **evBringUp:** Indica externamente que a negociação da canal de controle deve começar. Este evento, por exemplo, pode ser provocado pelo comando do operador, pela conclusão bem sucedida de um procedimento *bootstrap* do canal de controle, ou pela configuração. Dependendo dessas configurações, isto provocará

1a) a emissão de uma mensagem Config,

```
ret_send ← chama a função para enviar a mensagem Config no modo multicast
Se ret_send igual a 1 // Se enviou corretamente retorna 1
    ESTADO ←ConfSnd
    Parada
Fim Se
```

1b) um período da espera para receber uma mensagem Config do nó remoto.

```
Enquanto 1
    Faça
    sockc ← recebe mensagem Config do vizinho
    Se sockc diferente de erro faça
        Se mensagem recebida igual a Config faça
        ESTADO ← ConfRcv
        Parada
        Fim Se
    Fim Se
 Fim
```

2: **evCCDn**: Este evento é gerado quando há uma indicação que a canal do controle está já não disponível. Força o MEF sempre ao estado *down*.

Início

Fecha (sock) //socket

Fecha (socke) //socket

HELLO_INTERVAL ← 500;

HELLO_DEAD_INTERVAL ← 1500;

TxSeqNum← 1

RcvSeqNum←0;

Chama função de gerenciamento_canal

Parada

Fim

3: **evConfDone:** Este evento indica que uma mensagem de ConfigAck foi recebida, reconhecendo os parâmetros dos Config.

Se sockc igual a ConfigACK faça

ret_send ← chama função para enviar uma ConfigAck //aceitou os parâmetros

ESTADO ←Active

contador ← 0

relogio ← 0

Parada

Fim Se

4: **evConfErr:** Este evento indica que uma mensagem de ConfigNack foi recebida, rejeitando os parâmetros dos Config.

Se sockc igual a ConfigNACK faça

neg ← chama função de negociação dos parâmetros

Se neg igual a ok faça//14 reconfigura

ret_send ← chama função para enviar uma Config com os novos parâmetros

ESTADO ← ConfSnd

Parada

Fim Se

Senão

ret_send ← chama função para enviar uma ConfigNack com os novos parâmetros

ESTADO ← ConfSnd

Parada

Fim Se

```
Contador←0
Relógio←0
Parada
Fim Se
```

5: **evNewConfOK**: A nova mensagem Config foi recebida do vizinho e reconhecida positivamente.

```
rp ← chama função que reconhece parâmetros da mensagem Config
Se rp igual a 1faça //Aceito os parâmetros
    escreva("enviar ack\n")
    ret_send ← chama função enviar ConfigAck
    escreva("Envia Config Ack\n")
ESTADO←Up
Contador←0
Relógio←0
Parada
Fim Se
```

6: **evNewConfErr:** A nova mensagem Config foi recebida do vizinho e rejeitada com uma mensagem ConfigNack.

```
Senão Se parametros da mensgaem Config igual a rejeitados faça
    ret_send ← chama função para enviar ConfigNack
    ESTADO←Down
    Parada
Fim Se
```

7: **evContenWin:** A nova mensagem Config foi recebida do vizinho, e ao mesmo tempo uma mensagem Config foi emitida ao vizinho. O nó local ganha a disputa. Com o resultado, a mensagem recebida Config é ignorada.

```
Se tempo da mensagem recebida igual a tempo faça   //nó local ganha disputa
  result1← NODEID
  result2 ← NODEIDRCV
   Se result1 menor que result2 faça
      //Ignora Config recebida;
      ESTADO ← ConfSnd;
      Parada
       Fim Se
Fim Se
```

8: **evContenLost:** A nova mensagem Config foi recebida do vizinho, e ao mesmo tempo uma mensagem Config foi emitida ao vizinho. O nó local perde a disputa.

8a) A mensagem Config é positivamente reconhecida.

8b) A mensagem Config é negativamente reconhecida.

```
Se result1maior que result2
      rp ← chama função reconheceparametros da mensagem Config
            Se rp igual a1 faça//aceito parâmetros
                  escreva("enviar ack\n")
                  ret_send ← chama função para enviar ConfigAck
                  escreva("Envia Config Ack\n")
                  ESTADO← Up
                  Contador←0
                  Relógio←0
                  Parada;
        Fim Se
          Senão //NACK
                  ESTADO ← ConfRcv
                  Contador←0
                  Relógio←0
                  Parada
            Fim Se
Fim Se
```

9: **evAdminDown:** O administrador pede que o canal de controle vá para down.

```
lmp.flag ← "0x01"//Ativa o Flag que indica para o canal ir para Down
ESTADO← GoingDown
Parada
```

10: **evNbrGoesDn**: Um pacote com o *flag* ControlChannelDown é recebido do vizinho.

Se (comparação entre lmprcv.flag e "0x01") igual a 0 faça //Recebeu um pacote com o Flag que indica que o canal foi para Down

//10 - Um pacote com o flag ControlChannelDown Ã© recebido do vizinho.

Para cont=0 até cont<550 repita

fecha (sock)

fecha (socke)

HELLO_INTERVAL ← 500

HELLO_DEAD_INTERVAL ←1500

TxSeqNum← 1

RcvSeqNum←0

Chama função de gerenciamento_canal

Parada

Fim Para

Fim Se

11: **evHelloRcvd:** Um pacote Hello com o esperado SeqNum foi recebido.

Se sockc recebe uma mensagem Hello faça

TxSeqNum←TxSeqNum+1

Se RcvSeqNum igual a TxSeqNum

Para HELLO_INTERVAL=1 até HELLO_INTERVAL!=0 decremente

ret_send ←chama função para enviar mensagens Hello

//11- Um pacote Hello com o esperado SeqNum foi recebido.

ESTADO ←Up

Parada

Fim Para

Fim Se

12: **evHoldTimer**: O tempo do HelloDeadInterval expira indicando que nenhum pacote Hello foi recebido. Isto move o canal de controle novamente na negociação do estado, e dependendo da configuração local, isto provocará:

12a) a emissão de mensagens Config periódicas,

12b) um período da espera para receber mensagens Config do nó remoto.

Se HelloDeadInterval igual a 0 faça

//12b **Enquanto contador diferente de 0**

Se mensagem recebida for igual a mensagem Config faça

{...}

//12a **Senão ret_send ← chama função para enviar uma Config**//1ª restransmitido

decrementa relógio

Parada

Fim Se

Fim Enquanto

Fim Se

13: **evSeqNumErr:** Hello com o inesperado SeqNum recebido e rejeitado.

Se RcvSeqNum diferente de TxSeqNum+1 faça

ESTADO←Up

Decrement HelloDeadInterval

Decrementa HelloInterval

Fim Se

14: **evReconfig:** Parâmetros do canal de controle foram reconfigurados e requerem a renegociação.

Se neg igual a ok faça //14 reconfigura

ret_send ← chama função para enviam a nova mensagem Config com novos parâmetros

ESTADO ←ConfRcv

Parada

Fim Se

15: **evConfRet:** O tempo de retransmissão expirou e uma mensagem Config reenviada.

//12b **Enquanto contador diferente de 0**

{....}

Decremneta relogio

Parada

ret_send ← chama função para enviar uma Config

Fim

16: **evHelloRet:** O tempo de HelloInterval expirou e o pacote Hello e é enviado.

Se HelloInterval igual a 0 faça
ret_send ← chama função para enviar mensagem Hello
ESTADO←Up
Parada
Fim Se

17: **evDownTimer:** O tempo expirou e nenhuma mensagem foi recebida com o *flag* ControlChannelDown ajustado.

Enquanto contador diferente de 0
Se (comparação entre lmp.flag e "0x01") igual a 0 faça
ESTADO ← GoingDown
Parada
Fim Se
Decrementa contador
ESTADO ← Down
Fim

3.3.2. MEF do Enlace TE

A MEF do enlace TE define os estados e as lógicas da operação do enlace TE do LMP.

Estados do Enlace TE

A correlação das propriedades de enlace em um único enlace TE pode estar em um dos estados descritos abaixo, conforme mostra a Figura 20. Cada estado corresponde a uma determinada condição do enlace TE e é associado geralmente com um tipo específico de mensagem de LMP que é transmitida periodicamente à extremidade distante através do canal de controle associado ou através dos enlaces de dados.

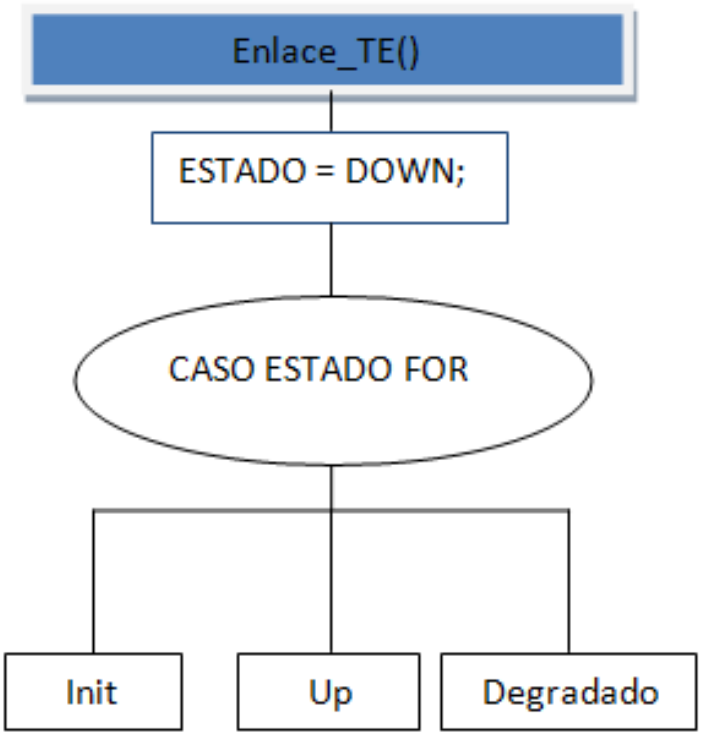


Figura 20 – Fluxograma do Enlace TE.

Down: Não há nenhum enlace de dados alocado ao enlace TE.

Init: Os enlaces de dados foram alocados ao enlace TE, mas a configuração ainda não foi sincronizada com o LMP vizinho. A mensagem de LinkSummary é periodicamente transmitida ao vizinho de LMP.

Up: Este é o estado normal operacional do enlace TE. No mínimo um canal de controle LMP é solicitado para ser operacional entre os nós que compartilham o enlace TE. Como parte da operação normal, a mensagem de LinkSummary pode ser transmitida periodicamente ao vizinho LMP ou gerado por um pedido externo.

Degradado: Neste estado, todos os canais de controle de LMP estão down, mas o enlace TE inclui ainda alguns enlaces de dados que são alocados ao tráfego do usuário.

Eventos do enlace TE

Na função de correlação das propriedades dos enlaces, a função Enlace_TE() é chamada. A mensagem m_LinkSummary() é usada para sincronizar o Interface_Ids e correlacionar as propriedades do enlace TE. A mensagem LinkSummary pode ser trocada a qualquer momento, mesmo se um enlace não estiver em processo de verificação. A mensagem LinkSummary deve ser periodicamente transmitida até que o nó receba uma m_LinkSummaryAck() ou mensagem m_LinkSummaryNack() ou um limite de repetição for atingido. Ambos os intervalos de retransmissão e de limite de repetição são parâmetros locais de configuração.

A mensagem m_LinkSummaryAck só será recebida se os parâmetros contidos no Data_Link forem correlacionados. Caso contrário o enlace será considerado não correlacionado, portanto não será agregado ao Enlace_TE.

Conforme mencionado acima, a função de correlação de enlace TE deveria vir depois da verificação da conectividade do enlace, já que parâmetros como o Link_ID e o Interface_ID devem ser conhecidos antes da correlação, más só são apresentados na verficação de conectividade do enlace. Portanto, será considerado apenas como uma base para futuramente implementar a verificação de conectividade e testar suas funcionalidades.

Assim como na operação do canal do controle, a operação do enlace TE de LMP é descrita nos termos da MEF em estados e eventos. Eventos são gerados pelos processamentos das rotinas do pacote e pelos MEFs associados aos canais de controle e enlace de dados. A Figura 21 ilustra a operação do MEF do enlace TE em forma de diagrama da transição do estado MEF.

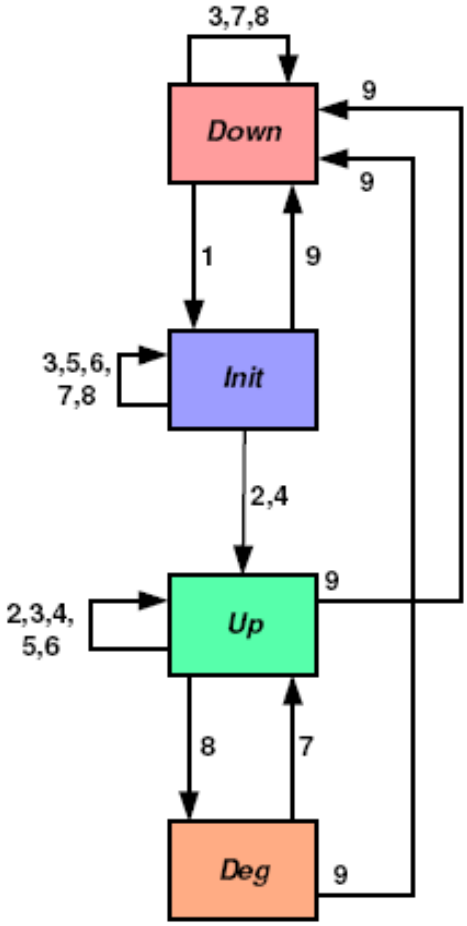


Figura 21 – MEF do Enlace LMP-TE (CUNHA, 2006).

Os eventos responsáveis para integrar esses enlaces são descritos à seguir.

1: **evDCUp:** Um ou mais canais dos dados foram habilitados e atribuídos ao enlace TE.

Se ESTADO igual a Active faça

ESTADO2 ← Up

Parada

Fim Se

2: **evSumAck:** Mensagem LinkSummary recebida e positivamente reconhecida.

```
Se sockc recebeu uma mensagem linkSummary faça
    neg ← chama função que reconhece parâmetros de TE
    Se correlacionou os parâmetros
        ret_send ←chama função para enviar uma mensagem LinkSummaryAck
        ESTADO2 ← Up
        contador ←0
        relogio ← 0
        Parada
    Fim Se
Fim Se
```

3: **evSumNack:** Mensagem LinkSummary recebida e negativamente reconhecida.

```
Se sockc recebeu uma mensagem linkSummary faça
    neg ← chama função que reconhece parâmetros de TE
    Se correlacionou os parâmetros faça {...}
    Senão ret_send ← chama função para enviar uma LinkSummaryNack
        ESTADO2 ← Init
        contador ←0
        relogio ←0
        Parada
    Fim Se
Fim Se
```

4: **evRcvAck:** Mensagem LinkSummaryAck recebida reconhecendo a configuração do enlace TE.

```
Se sockc recebeu uma mensagem de LinkSummaryAck faça // reconhecendo os parâmetros
        ret_send ← chama função para enviar uma LinkSummaryAck
        contador ← 0
        relogio ← 0
        i ← i + 1
        EnlaceTE[i] ←INTERFACEID
        enlace ← i
        ESTADO2 ← Up
        Parada
Fim Se
```

5: **evRcvNack:** Mensagem de LinkSummaryNack recebida.

```
Se sockc recebeu uma mensagem de LinkSummaryAck faça
    neg ← chama função de negociação dos parâmetros
    Se neg igual a ok faça //paraâmetros aceitos
        ret_send ← chama função para enviar uma LinkSummary com os novos parâmetros
        ESTADO ← Up
        Parada
    Fim Se
    Senão
        ret_send ← chama função para enviar uma LinkSummaryNack
        ESTADO ← Down
        Parada
    Fim Se
    Contador←0
    Relógio←0
    Parada
Fim Se
```

6: **evSumRet:** O tempo de retransmissão expirou e a mensagem LinkSummary é enviada.

```
contador ← contador * (1 + delta)
relógio←100
Enquanto relógio diferente de 0
    Enquanto contador diferente de 0
        Se relógio igual a 0 faça
            ret_send ← chama função para enviar uma LinkSummary
        Senão
            Decrementa relógio
        Fim Se
    Fim
Fim
```

7: **evCCUp:** O primeiro canal de controle ativo vai para *up*.

```
Se ESTADO igual a Up faça
        ESTADO2←Up
Fim Se
```

8: **evCCDown:** Último canal de controle ativo o controle vai para *down*.

SenãoSe ESTADO igual a Down faça
ESTADO2← Init
Fim Se

9: **evDCDown:** Último canal de dados do enlace TE foi removida.

Senão ESTADO2← Down

3.3.3. Mensagens LMP

As mensagens LMP têm em comum o cabeçalho LMP apresentado na Figura 22 a seguir. Esse cabeçalho somente pode, em alguns casos, ser limitado nas mensagens de teste.

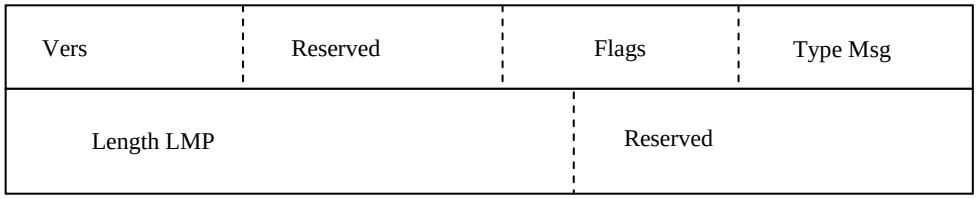


Figura 22 – Cabeçalho LMP.

O campo Reservado deve ser emitido com valor zero e ignorado no recebimento. Já o campo Vers é a versão do protocolo que no caso é 1. Os *flags* indicam se o canal vai para Down "forçadamente" (0x01), ou indica seu reinício (0x02).

O *Type Msg* pode ser ajustada indicando a mensagem enviada/recebida, conforme abaixo:

1 = Config
2 = ConfigAck
3 = ConfigNack
4 = Hello
5 = BeginVerify
6 = BeginVerifyAck
7 = BeginVerifyNack

8 = EndVerify

9 = EndVerifyAck

10 = teste

11 = TestStatusSuccess

12 = TestStatusFailure

13 = TestStatusAck

14 = LinkSummary

15 = LinkSummaryAck

16 = LinkSummaryNack

17 = ChannelStatus

18 = ChannelStatusAck

19 = ChannelStatusRequest

20 = ChannelStatusResponse

Todas as mensagens são emitidas sobre o canal de controle, exceto a mensagem de Teste que é emitida sobre o enlace de dados que está sendo testado.

As mensagens de LMP são construídas usando objetos, os quais possuem os parâmetros que são utilizados nas fases de negociação. Cada objeto é identificado por sua classe (*C-type*), e cada classe tem um nome (*Class*). Os objetos de LMP podem ser negociáveis ou não-negociáveis (identificado pelo *bit* de N), conforme mostra a Figura 23. Se não negociáveis, não são avaliados simplesmente considerados.

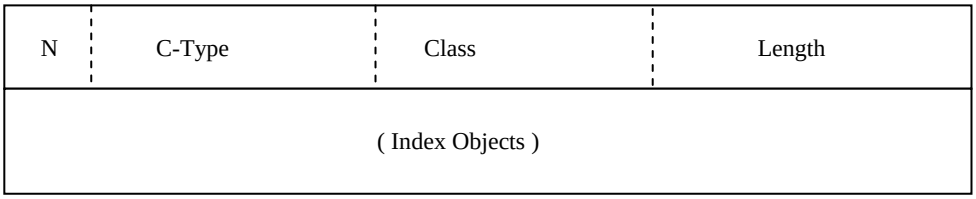


Figura 23 – Objetos LMP.

Dentro do índice de objetos (*Index Objects*) são enviados os parâmetros em seqüência, conforme a classe que representa o tipo de mensagem enviada/recebida no cabeçalho LMP. Abaixo as mensagens utilizadas no programa serão detalhadas:

Mensagem de Config (tipo dos Msg = 1)

A mensagem de Config é usada na fase da negociação do canal de controle. Os índices da mensagem dos Config são construídos usando os seguintes objetos LMP:
<Mensagem dos Config>::= <cabeçalho LMP> <LOCAL_CCID> <MESSAGE_ID> <LOCAL_NODE_ID> <CONFIG>

```
Programa m_Config
    Início
    ret_send,rs1←0, rs2←0, rs3←0: inteiro
    lmp.msg_classe ←1 //1 = Classe Config
    chama função CCID
     aguarda 1s
    rs1← chama função MESSAGE_ID
    aguarda 1s
    rs2← chama função NODE_ID
    aguarda 1s
    rs3← chama função CONFIG
    aguarda 1s
    escreva("m_config")
    Se (rs1 igual a 5) ou (rs2 igual a 5) ou (rs3 igual a 5) faça
        ret_send ←5
    Senão
        ret_send←1
    Fim Se
Escreva ("%d", ret_send)
retorne ret_send
 Fim
```

Mensagem de ConfigAck (tipo dos Msg = 2)

A mensagem de ConfigAck é usada para reconhecer o recebimento de uma mensagem Config e indica o acordo em todos os parâmetros. Os índices da mensagem de ConfigACK são construídos usando os seguintes objetos LMP:
<Mensagem de ConfigAck>:: = <cabeçalho LMP> <LOCAL_CCID> <LOCAL_NODE_ID> <REMOTE_CCID> <MESSAGE_ID_ACK>< REMOTE_NODE_ID>

```
Programa m_ConfigAck
    Início
    ret_send, rs1←0, rs2←0, rs3←0, rs4←0, rs5←0 : inteiro
    lmp.msg_classe ←2              //2 = Classe ConfigAck
    lmp.obj.classe←1
    lmp.obj.c_type ←1
    lmp.obj.n ← 0
    lmp.obj.objeto[1] ← CCIDV
    escreva ("CCID%ld\n",lmp.obj.objeto[1])
    chama função para enviar um socket
    rs1←1
    aguarda 1s
    rs2 ←chama função NODE_ID
    aguarda 1s
    rs3 ← chama função CCID remoto
    aguarda 1s
    rs4 ← chama função MESSAGE_ID remoto
    aguarda 1s
    rs5 ← chama função NODE_ID remoto
    aguarda 1s
    Se (rs1 igual a 5)ou(rs2 igual a 5)ou (rs3 igual a 5) ou (rs4 igual a 5) ou (rs5 igual a 5) faça
        ret_send ← 5
    Senão
        ret_send← 1
    Fim Se
    escreva("m_ConfigAck")
    retorne ret_send
  Fim
```

Mensagem de ConfigNack (tipo dos Msg = 3)

A mensagem de ConfigNack é usada para reconhecer o recebimento de uma mensagem Config. Porém, não concordando com seus parâmetros, se esses parâmetros indicarem que são negociáveis, deve-se propor outros valores. Os índices da mensagem dos ConfigNACK são construídos usando os seguintes objetos LMP:

<Mensagem de ConfigNack> :: = <cabeçalho LMP><LOCAL_CCID> <LOCAL_NODE_ID> <REMOTE_CCID><MESSAGE_ID_ACK><REMOTE_NODE_ID> <CONFIG>

Programa m_ConfigNack

ret_send,rs1←0, rs2←0, rs3←0, rs4←0, rs5←0, rs6←0: inteiro

lmp.msg_classe ←3 //3 = Classe ConfigNack

lmp.obj.classe←1

lmp.obj.c_type ←1

lmp.obj.n ← 0

lmp.obj.objeto[1] ← CCIDV

escreva("CCID%ld\n",lmp.obj.objeto[1])

chama função de envio do socket

rs1←1

aguarda 1s

rs2 ← chama função NODE_ID

aguarda 1s

rs3 ← chama função CCID remoto

aguarda 1s

rs4 ← chama função MESSAGE_ID remoto

aguarda 1s

rs5← chama função NODE_ID

aguarda 1s

rs6 ← chama função CONFIG

aguarda 1s

Se (rs1 igual a 5) ou (rs2 igual a 5) ou (rs3 igual a 5) ou (rs4 igual a 5) ou (rs5 igual a 5) ou (rs6 igual a 5) faça

ret_send ←5

Senão

ret_send←1

Fim Se

retorne ret_send

Fim

Se a mensagem de ConfigNack incluir objetos não negociáveis, seus parâmetros devem ser copiados dentro da mensagem e reenviado.

Hello mensagem (tipo dos Msg = 4)

Os índices da mensagem dos Hello são construídos usando os seguintes objetos LMP:
<Hello mensagem> :: = <cabeçalho LMP> <LOCAL_CCID> <HELLO>

```
Programa m_Hello
    Início
    ret_send, rs1←0, rs2←0 : inteiro
    lmp.msg_classe=←4              //4 = Classe Hello
    lmp.obj.classe←1
    lmp.obj.c_type ←1
    lmp.obj.n ← 0
    lmp.obj.objeto[1] ← CCIDV
    escreva ("CCID%ld\n",lmp.obj.objeto[1])
    chama função de envio do socket
    rs1←1
    aguarda 1s
    rs2 ← chama função Hello
    aguarda 1s
    Se (rs1 igual a 5) ou (rs2 igual a 5) faça
        ret_send ←5
    Senão
        ret_send←1
    Fim Se
    printf("m_Hello")
    retorne ret_send
Fim
```

Mensagem de LinkSummary (tipo dos Msg = 14)

A mensagem de LinkSummary é usada para sincronizar o Interface_Ids e correlacionar as propriedades do enlace de TE. Os índices da mensagem dos LinkSummary são construídos usando os seguintes objetos LMP:
<Mensagem de LinkSummary> :: = <encabeçamento comum> <MESSAGE_ID> <TE_LINK> <DATA_LINK> [<DATA_LINK>…]

```
Programa m_LinkSummary
    Início
    ret_send, ct1, ct2, rs1←0, rs2←0, rs3←0 : inteiro
    ct1 ← chama função ctype(11,0)
    ct2 ← chama função ctype(12,0)
    lmp.msg_classe ← 14              //14 = classe LinkSummary
    rs1 ← chama função MESSAGE_ID
    aguarda 1s
    rs2 ← chama função TE_LINK
    aguarda 1s
    rs3 ← chama função DATA_LINK             // tipo wavelength
    Se (rs1 igual a 5) ou (rs2 igual a 5) ou (rs3 igual a 5) faça
        ret_send =5
    Senão
        ret_send←1
    Fim Se
    escreva("m_LinkSummary")
    retorne ret_send
Fim
```

Mensagem de LinkSummaryAck (tipo dos Msg = 15)

A mensagem de LinkSummaryAck é usada para indicar o acordo com a sincronização e aceitação/acordo dos parâmetros de enlace de Interface_Id. Os índices da mensagem dos LinkSummaryAck são construídos usando os seguintes objetos LMP:

<Mensagem de LinkSummaryAck> :: = < cabeçalho LMP > <MESSAGE_ID_ACK>

```
Programa LinkSummaryAck
    Início
    ret_send, rs1←0 : inteiro
    lmp.msg_classe ← 15                  //15 = Classe LinkSummaryAck
    rs1 ← chama função MESSAGE_ID
    Se (rs1 igual a 5) faça
        ret_send ←5
    Senão
        ret_send←1
    Fim Se
    retorne ret_send
Fim
```

Mensagem de LinkSummaryNack (tipo dos Msg = 16)

A mensagem de LinkSummaryNack é usada para indicar sobre o que os parâmetros não foram aceitos e propor outros valores. Os índices da mensagem dos ConfigNACK são construídos usando os seguintes objetos LMP:

<Mensagem de LinkSummaryNack> :: = < cabeçalho LMP > <MESSAGE_ID_ACK> <ERROR_CODE> [<DATA_LINK>…]

```
Programa m_LinkSummaryNack
    Início
    ret_send, rs1←0 : inteiro
    lmp.msg_classe ← 16              // 16 = Classe LinkSummaryNack
    rs1 ←  chama função MESSAGE_ID
    Se (rs1 igual a 5) faça
          ret_send ←5
    Senão
          ret_send←1
    Fim Se
    escreva ("m_LinkSumaryNack")
    retorne ret_send
Fim
```

Se a mensagem de LinkSummaryNack incluir objetos não negociáveis, seus parâmetros devem ser copiados dentro da mensagem e reenviados.

Se a mensagem for recebida, porém o C-Tipo for desconhecido, uma mensagem ERROR_CODE deve indicar o erro.

3.3.4. Definições do objeto de LMP

Os objetos passados como parâmetros são definidos utilizando a classe o *c_type*, conforme descritos abaixo:

Classe CCID

Apresenta a identificação (ID) do canal de controle. Sua classe é representada pelo número 1, possui 32 *bits* de tamanho, e sua forma no programa é demonstrada abaixo:

C-Tipo = 1, LOCAL_CCID/C-Tipo =2 REMOTE_CCID

```
Programa CCID
    Início
    // Identificador do canal de controle
    REMOTE_CCID, CCIDV2←1 : inteiro longo
    lmp.obj.classe ← 1
    Faça caso c_type
        caso 1:   CCIDV← CCIDV2
                  lmp.obj.c_type ←c_type
                  lmp.obj.n ← 0
                  lmp.obj.objeto[1] ← CCIDV
                  escreva ("CCID%ld\n",lmp.obj.objeto[1])
                  chama função de envio do socket
                  incrementa CCIDV2
                  retorne 1
                  parada
        case 2:   lmp.obj.c_type ← c_type
                  REMOTE_CCID ← CCIDRCV
                  lmprcv.objrcv.objeto[1] ← REMOTE_CCID
                  escreva ("REMOTE_CCID%ld\n",REMOTE_CCID)
                  lmprcv.objrcv.n ← 0
                  chama função de envio do socket
                  retorne 1
                  parada
    Fim Faça
    Chama função de ERROR_CODE
    retorne 5
Fim
```

Este objeto é não negociável.

Classe NODE_ID

Apresenta a identificação (ID) do nó que originou o pacote, e do nó remoto. Sua classe é representada pelo número 2, possui 4 *Bytes* de tamanho, e sua forma no programa é demonstrada abaixo:

C-Tipo = 1, LOCAL_NODE_ID/ C-Tipo =2 REMOTE_NODE_ID

```
Programa NODE_ID
    Início
    REMOTE_NODE_ID : inteiro longo
    lmp.obj.classe ← 2
    incrementa NODEID
    Faça caso c_type
        case 1:lmp.obj.c_type ← c_type
            lmp.obj.objeto[1] ← NODEID
            escreva ("NODEID%ld\n",lmp.obj.objeto[1])
            chama função de envio do socket
            lmp.obj.n ← 0
            retrono 1
            parada
        case 2:lmp.obj.c_type ←c_type
            REMOTE_NODE_ID ←NODEIDRCV          //recebe node id
            lmprcv.objrcv.objeto[1]← REMOTE_NODE_ID
            lmprcv.objrcv.n ← 0
            chama função de envio do socket
            retorna 1
            parada
        Fim Faça
        Chama função de ERROR_CODE
    retorne 5
Fim
```

Este objeto é não negociável.

Classe MESSAGE_ID

Apresenta a identificação (ID) da mensagem. Este valor é sempre incrementado depois que enviada a mensagem e somente decrementado com seu estouro ou reinício. Sua classe é representada pelo número 5, possui 4 *Bytes* de tamanho, e sua forma no programa é demonstrada abaixo:

C-Type=1, MessageId/ C-Tipo =2 MessageIdAck

```
Programa MESSAGE_ID
    Início
    MESSAGEID, MESSAGEIDRCV : inteiro longo
    lmp.obj.classe ← 5
    incrementa MESSAGEID
    MESSAGEIDRCV← MESSAGEID + 1
    Se c_type igual a 1faça //IPV4
        lmp.obj.c_type ← c_type
        lmp.obj.objeto[1] ← MESSAGEID
        lmp.obj.n ← 0              //não negociáveis
        escreva ("MESSAGEID%ld\n",lmp.obj.objeto[1])
        chama função de envio do socket
        retorna 1
    Senão Se c_type igual a 2 faça//IPV6
        lmp.obj.c_type ← c_type;
        lmp.obj.objeto[1] ← MESSAGEIDRCV
        lmp.obj.n ← 0
        chama função de envio do socket
        retorna 1
    Fim Faça
    Chama função de ERROR_CODE
    retorne 5
Fim
```

Este objeto é não negociável.

Classe CONFIG

A classe Config, representada pelo número 6, possui como parâmetros os tempos HelloInterval e HelloDeadInterval medidos em milissegundos (ms), que são enviados dentro da mensagem Config. Cada valor têm 16 *bits* de tamanho. Se o mecanismo *keep-alive* (manter-ativo) não for usado, esses valores devem ser ajustados em zero.

C-Tipo = 1, HelloConfig (HelloInterval, HelloDeadInterval)

```
Programa CONFIG
    Início
    i : inteiro
    HelloConfig[2] : inteiro longo
    Escreva ("CONFIG\n")
    Para i=0 até i>=2 repita
    HelloConfig[i]←'0'
    lmp.obj.classe ← 6
    Se c_type igual a 1 faça  //IPV4
        lmp.obj.c_type ← c_type
        HelloConfig[0]← HELLO_INTERVAL
        Escreva ("%ld ",HelloConfig[0])
        HelloConfig[1] ← HELLO_DEAD_INTERVAL
        Escreva ("%ld ",HelloConfig[1])
        lmp.obj.n ← 1
        para i=0 até i<2 repita
            lmp.obj.objeto[i]←HelloConfig[i]
            escreva ("%ld \n",lmp.obj.objeto[i])
        Fim Para
         chama função de envio do socket
    Fim Faça
    Chama função de ERROR_CODE
    retorne 5
Fim
```

Esse objeto é negociável, se tiverem valores diferentes adota-se o maior.

Classe HELLO

Hello representada pelo número 7, possui como parâmetros os tempos TxSeqNum (Número de Seqüência da Mensagem Enviada) e RcvSeqNum (Número de Seqüência da Mensagem Recebida), que identificam se nenhuma mensagem foi perdida. Cada valor têm 32 *bits* de tamanho. O valor inicial de RcvSeqNum é 0 e de TxSeqNum é 1, e devem permanecer com a diferença de 1.

C-Tipo = 1, Hello (TxSeqNum, RcvSeqNum)

```
Programa Hello
    Início
    HELLO_G[2] : inteiro longo
    i : inteiro
    lmp.obj.classe ← 7
    libera memória HELLO_G
    para i=0 até i>=2 repita
    HELLO_G[i]← '0'
    Se c_type igual a 1 faça          //IPV4
        lmp.obj.c_type ←c_type
        Se RcvSeqNum diferente de 0 faça
            TxSeqNum ← TxSeqNum + 1
            HELLO_G[0] ← TxSeqNum
            HELLO_G[1] ← RcvSeqNum
            Se TxSeqNum igual a 0
                ESTADO ←Down
            Fim Se
        Se HELLO_G[0] igual a HELLO_G[1] faça
            Compia "0" em lmp.flag
            TxSeqNum← 1
            RcvSeqNum←0
            HELLO_G[0] ← TxSeqNum
            HELLO_G[1] ← RcvSeqNum
            ESTADO ← 0
        Fim Se
        Se HELLO_G[0] igual a 4294967295 faça          //((2^32)-1))
            TxSeqNum ←2
            HELLO_G[0]←TxSeqNum
        Fim Se
```

```
Fim Se
Senão
        TxSeqNum=←1          //Calculo inicial;
        RcvSeqNum←0          // Indica que nenhum Hello foi recebido
        HELLO_G[0] ←  TxSeqNum
        HELLO_G[1] ← RcvSeqNum
        lmp.obj.n ← 0
Fim Se
Para i=0 até i<2 repita
        lmp.obj.objeto[i]← HELLO_G[i]
        escreva ("HELLO%ld ",lmp.obj.objeto[i])
Fim Para
chama função de envio do socket
escreva ("exit hello");
retorne 1
chama função de ERROR_CODE
retorne 5
Fim
```

Este objeto é não negociável.

Classe TE_LINK

A classe TE_Link referenciado pelo número11, possui como parâmetros os campos Flag (8*bits*), Local_Link_ID e Remote_Link_ID, apresentado na Figura 24. O campo flag identifica qual funcionalidade dessa classe: se 0x01, irá suportar gerência de falha; se 0x02, suporta verificação do enlace. Já os campos Local_Link_ID e Remote_Link_ID, representam os Ids dos enlaces, devem ser diferentes de zero e possuem tamanhos diferentes, pois suportam IPV4 (4 *bytes*), IPV6 (16 *bytes*), além dos Sem Numeração com tamanho máximo de 16 *bytes*.

C-Tipo = 1, IPv4 TE_LINK/ C-Tipo = 2, IPv6 TE_LINK/ C-Tipo = 3, TE_LINK *Unnumbered*

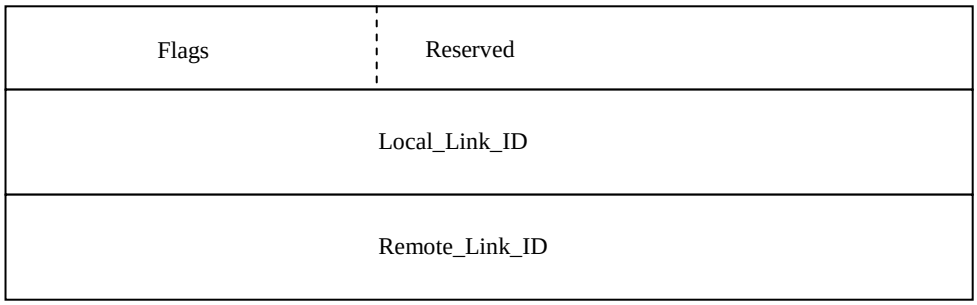


Figura 24 – Cabeçalho TE_LINK.

Programa TE_LINK

Início

tipo[10] : char

Estrutura te

Início

flag_tl[4] : char

Local_Link_Id : inteiro longo

Remote_Link_Id : inteiro longo

reserved_tl : inteiro

Fim Estrutura

IPV4_TE_LINK : inteiro longo

IPV6_TE_LIN : inteiro longo

TE_LINK_UNNUMBERED : inteiro longo

lmp.obj.classe ←11

incrementa te.Local_Link_Id

incrementa te.Remote_Link_Id

copia,"0x00" em te.flag_tl // nao gerência de falha suportada.

Se comparação entre te.flag_tl e "0x02" forem iguais faça //nao suporta verificacao de enlace

Faça caso c_type

case 1:lmp.obj.c_type ← c_type

IPV4_TE_LINK ← te.Local_Link_Id

lmp.obj.objeto[1] ← IPV4_TE_LINK

chama função de envio do socket

retorna 1

parada

case 2:lmp.obj.c_type ← c_type

IPV6_TE_LINK ← te.Local_Link_Id

lmp.obj.objeto[1] ← IPV6_TE_LINK

chama função de envio do socket

retorna 1

parada

```
        case 3:lmp.obj.c_type ← c_type
                TE_LINK_UNNUMBERED ← te.Local_Link_Id
                 lmp.obj.objeto[1] ← TE_LINK_UNNUMBERED
                chama função de envio do socket
                retorna 1
                parada
        Fim Faça
    Chama função de ERROR_CODE
    retorne 5
Fim
```

Classe DATA_LINK

A classe Data_Link representada pelo número 12, possui como parâmetros os campos *Flag* (8*bits*), Local_Interface_ID e Remote_Link_ID e os *sub-objects*, apresentada na Figura TTTTT. O campo *flag* identifica qual funcionalidade dessa classe: se ajustado para 0x01, o enlace de tráfego é uma porta senão um; se ajustado para 0x02, é alocado para tráfego de usuário, e se ajustado para 0x04 o enlace de dados falhou e não está apropriado para tráfego de usuário. Já os campos Id_ Local_Interface_ID e Remote_Link_ID, representam os Ids das interfaces, devem ser diferentes de zero e possuem tamanhos diferentes, pois suportam IPV4 (4 *bytes*), IPV6 (16 *bytes*), além dos Sem Numeração com tamanho máximo de 16 *bytes*.

C-Tipo = 1, IPv4 DATA_LINK/ C-Tipo = 2, IPv6 DATA_LINK/ C-Tipo = 3, DATA_LINK *Unnumbered*

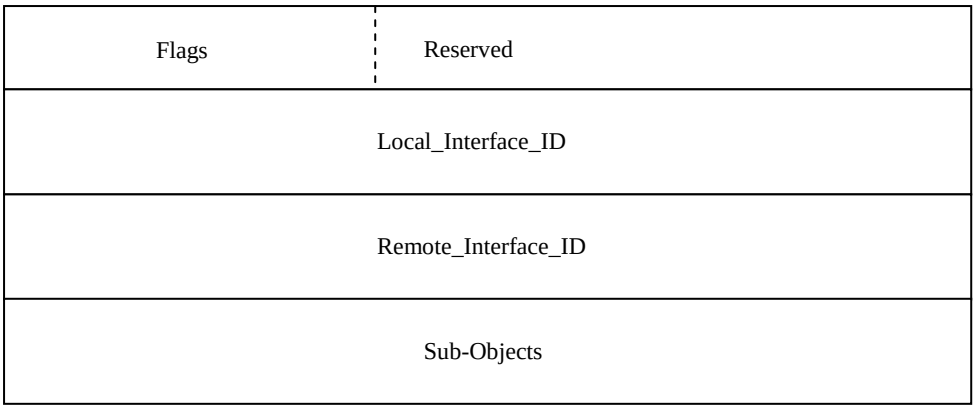


Figura 25 – Cabeçalho DATA_LINK.

Programa DATA_LINK

Início

tipo[10] : char

incrementa data.Local_Data_Id

IPV4_DATA_LINK : inteiro longo

IPV6_DATA_LINK : inteiro longo

DATA_LINK_UNNUMBERED : inteiro longo

Escreva ("entrou no data")

lmp.obj.classe ← 12

lmp.obj.n← 0

Faça caso c_type

case 1:lmp.obj.c_type ← c_type

IPV4_DATA_LINK ← data.Local_Data_Id

lmp.obj.objeto[1] ← IPV4_DATA_LINK

chama função de envio do socket

parada

case 2:lmp.obj.c_type ←c_type

IPV6_DATA_LINK ← data.Local_Data_Id

lmp.obj.objeto[1] ← IPV6_DATA_LINK

chama função de envio do socket

parada

case 3:lmp.obj.c_type ← c_type

DATA_LINK_UNNUMBERED ← data.Local_Data_Id

lmp.obj.objeto[1] ← DATA_LINK_UNNUMBERED

chama função de envio do socket

parada

Fim Faça

Chama função de ERROR_CODE

//**Se comparação entre data.flag_dl e "0x01" forem iguais faça** {/* ENLACE DE DADOS EH UMA PORTA*/}

//Senão Se **comparação entre data.flag_dl e "0x02" forem iguais faça** {/*ENLACE DE DADOS EH PARA TRAFEGO DE USUARIO*/}

//Senão Chama função de ERROR_CODE

data.sub_obj.type ← type

Faça caso data.sub_obj.type

case 1: lmp.obj.objeto[1] ← 1

chama função tipo1subobjects()

chama função de envio do socket

retorna 1

parada

```
                case 2: lmp.obj.objeto[1] ← 2
                        escreva ("%ld", lmp.obj.objeto[2])
                        chama função tipo2subobjects()
                        chama função de envio do socket
                        retorna 1
                        parade
            Fim Faça
            Chama função de ERROR_CODE
            retorne 5
Fim
```

O campo Sub-Objects consiste em uma série de dados, o qual cada tem um formulário, conforme mostra Figura 26:

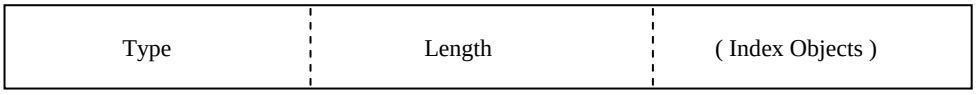


Figura 26 – Formulário *Sub-Objects.*

O tipo indicado pode ser 1 para relação de comutação e 2 para Wavelength.
Esses dados são definidos dentro do rótulo da LSP, portanto já no pacote DRAGON e só irão preencher os campos abaixo.

Tipo 1 de Sub-object: Tipo de Comutação

A RFC 3471 (BERGER, 2003), define os valores para o campo Type Switch (Tipo de Comutação) (8 *bits*) e o TypeEnc (8 *bits*), que é o tipo de codificação do enlace de dados. As larguras de faixa mínima e máxima (*Minimum/Maximum Reserved Bandwidth*) (32 *bits*) são medidas em *bytes* por segundo e representadas em ponto flutuante.

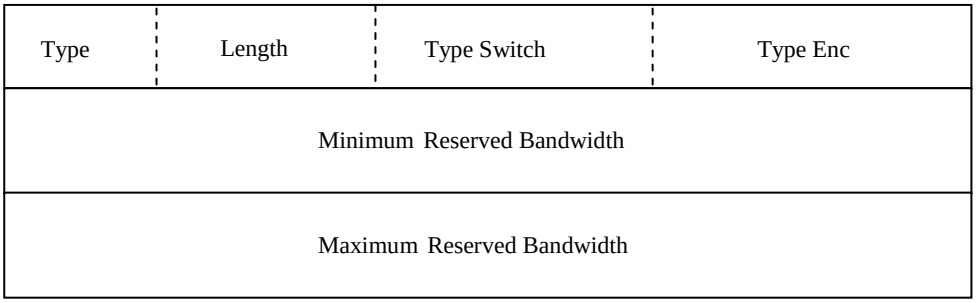


Figura 27 – Formulário Tipo 1 do *Sub-Object*.

Programa tipo1subobjects

Estrutura tipo1

type_so1: inteiro

length_so1: inteiro

Type_switchin : char

EncType_so1: char

Faixa_min : float

Faixa_max : float

Fim Estrutura

Fim

Tipo 2 de Sub-objeto: Wavelength

O campo Wavelength possui tamanho de 32 *bits* e também é definido pela RFC 3471.

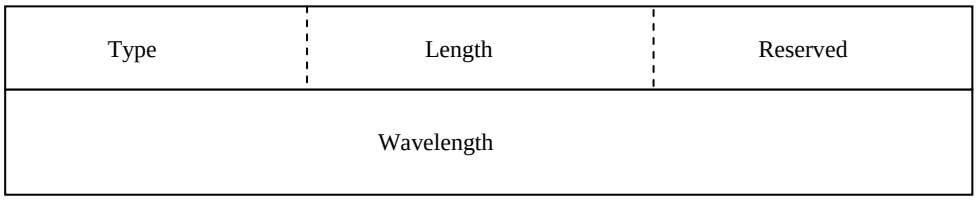


Figura 28 – Formulário Tipo 2 do *Sub-Object.*

programa tipo2subobjects

Estrutura tipo2

Type_so2: inteiro

int length_so2: inteiro

Reserved_so2 : char

Wavelength_so2 : inteiro longo

Fim Estrutura

Fim

Classe de ERROR_CODE

A classe 20 possui como parâmetros os retornos da mensagens que tiveram erro. Se C-Tipo for 1 são mensagens na fase de negociação se 2 são mensagens de erro do enlace TE.

C-Tipo = 1, BEGIN_VERIFY_ERROR

0x01 = Procedimento de Verificação do Enlace não suportado.

0x02 = Verificar Relutância.

0x04 = Mecanismo de Verificação de Transporte não suportado.

0x08 = Erro de Configuração de Link_Id.

0x10 = C-Tipo do objeto desconhecido.

C-Tipo = 2, LINK_SUMMARY_ERROR

0x01 = Parâmetros não negociáveis e inaceitáveis de LINK_SUMMARY.

0x02 = Renegociação do parâmetros de LINK_SUMMARY.

0x04 = Objeto inválido de TE_LINK.

0x08 = Objeto inválido de DATA_LINK.

0x10 = C-Tipo do objeto desconhecido de TE_LINK.

0x20 = C-Tipo do objeto desconhecido de DATA_LINK.

```
Programa ERROR_CODE(int c_type, char error[4]
    Início
    BEGIN_VERIFY_ERROR : char
    LINK_SUMMARY_ERROR : char
    lmp.obj.classe ← 20
    Se c_type igual a 1 faça
        Copia "BEGIN_VERIFY_ERROR" em error
        Se comparação entre error e "0x01" forem iguais
         //verifica enlace nao suportado
            Escreva ("Procedimento da verificação do enlace não suportado")
        Fim Se
        Se comparação entre error e "0x02" forem iguais
        //verificar Unwilling
            Escreva ("Verificar Unwilling")
        Fim Se
        Se comparação entre error e "0x04" forem iguais
        //Mecanismo nao suportado pela verificacao do transporte
            Escreva ("Mecanismo nao suportado pela verificacao do transporte ")
        Fim Se
        Se comparação entre error e "0x08" forem iguais
        //Configuracao do erro link_id
            Escreva ("O erro da configuração Link_Id")
        Fim Se
         Se comparação entre error e "0x10" forem iguais
        //Desconhecido C_type do objeto
            Escreva ("C-Tipo desconhecido do objeto")
        Fim Se
```

```
    Fim Se
    Se lmp.obj.c_type igual a 2
        Copia " LINK_SUMMARY _ERROR" em error
        Se comparação entre error e "0x01" forem iguais
        //Parametros nao aceitaveis e nao-negociaveis do LINK_SUMMARY
            Escreva ("Inaceitável parâmetros não-negociáveis de LINK_SUMMARY")
        Fim Se
        Se comparação entre error e "0x02" forem iguais
        //Renegociacao dos parametros do LINK_SUMMARY
            Escreva ("Renegociação de parâmetros LINK_SUMMARY")
        Fim Se
        Se comparação entre error e "0x04" forem iguais
        //Objeto invalido do TE_LINK
            Escreva ("0 objeto inválido de TE_LINK.")
        Fim Se
        Se comparação entre error e "0x08" forem iguais
        //Objeto invalido do DATA_LINK
            Escreva ("O objeto inválido de DATA_LINK.")
        Fim Se
        Se comparação entre error e "0x10" forem iguais
        //Desconhecido C_type do objeto TE_LINK
            Escreva ("C-Tipo desconhecido do objeto TE_LINK")
        Fim Se
        Se comparação entre error e "0x20" forem iguais
        //Desconhecido C_type do objeto DATA_LINK
            Escreva ("C-Tipo desconhecido do objeto TE_LINK")
        Fim Se
    Fim Se
Fim
```

Este objeto é não negociável.

Os códigos em linguagem C dos pseudo-códigos apresentados, estão no Apêndice A.

4. TESTES

Neste capítulo são mostrados alguns testes do funcionamento do protocolo LMP já adicionado ao pacote DRAGON, dando ênfase à funcionalidade de gerenciamente de canal de controle. Em seguida, apresentam-se testes de desempenho de rede que mostram a porcentagem em que esse pacote de controle consegue trafegar em uma rede congestionada.

4.1. Integração do LMP no DRAGON

A sinalização, roteamento e gerenciamento de enlaces são requisitos essenciais para um plano de controle completo do GMPLS. Neste contexto, o pacote DRAGON mostra-se fundamental, já que apresenta protocolos que atendem a dois desses requisitos: o RSVP responsável pela sinalização e o OSFP, que realiza o roteamento. Para a funcionalidade de sinalização, o protocolo RSVP que se encontra disponível no DRAGON. Já o *software* ZEBRA, que atua na funcionalidade de roteamento, foi adaptado ao DRAGON e executa o protocolo OSPF_TE e o RIP. Entretanto, para suprir a função de gerenciamento dos enlaces, o protocolo LMP foi incluído no pacote DRAGON.

O DRAGON pode ser encontrado no site http://dragon.east.isi.edu/twiki/bin/view /DRAGON/ControlPlane.

No Linux, o DRAGON se encontra no endereço: usr/local/dragon/bin, onde o arquivo dragon.sh estabelece a conexão e chama os executáveis do ZEBRA, do RSVP. Portanto, foi necessário modificar esse arquivo de inicialização, incluindo a chamada do LMP (APÊNDICE D – Documento Modificado). Os executáveis e outros programas utilizados no DRAGON estão na pasta usr/local/dragon/sbin.

O protocolo LMP possui duas partes: o gerenciamento do canal de controle (lmpcc) e correlação de enlaces TE (lmpete), executadas simultaneamente assim que o DRAGON é inicializado. A Figura 29 mostra a inicialização do DRAGON já contendo o pacote LMP.

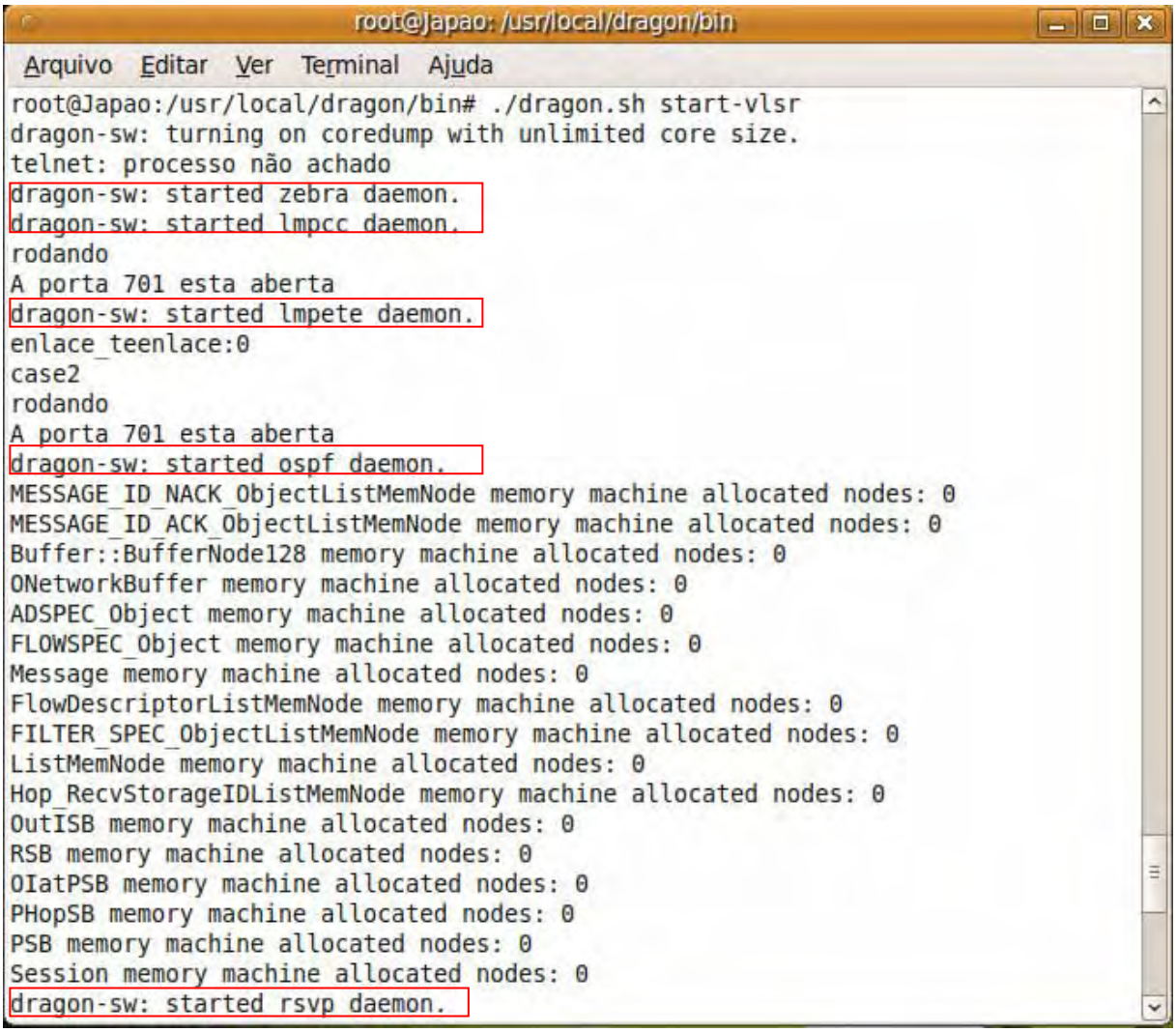


Figura 29 – Início o DRAGON.

A sequência de execução segue a ordem acima: primeiramente o protocolo OSPF no pacote ZEBRA, seguido do LMP com os pacotes lmpcc, ampliado na Figura 30, e lmpete, na Figura 31, logo o protocolo RSVP e por fim o daemon do DRAGON.

Dragon-sw: started lmpcc daemon.
Rodando
A porta 701 esta aberta

Figura 30 – Início do lmpcc.

Dragon-sw: started lmpete daemon.
enlace_teenlace:0
case2
rodando

Figura 31 – Início do lmpete.

Essa mesma sequência foi executada na parada manualmente do DRAGON mostrada na Figura 32.

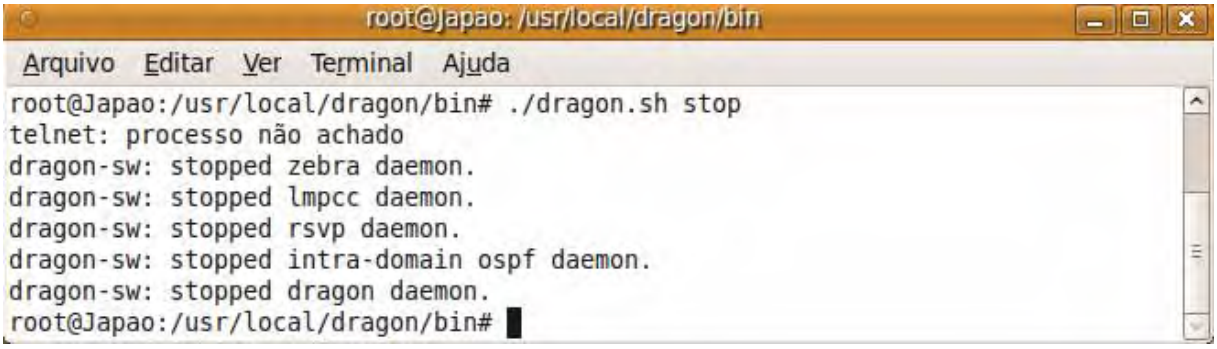


Figura 32 – Parada do DRAGON.

4.2. Testes de Funcionalidade

Os testes foram realizados no laboratório LACE (Laboratório de Automação e Computação Evolutiva) no Departamento de Ciências de Computação e Estatística (DCCE) da UNESP de São José do Rio Preto. Estes não puderam ser realizados em ambiente real de fibra, pois o projeto Kyatera, responsável pela entrega do sinal da fibra, ainda está com problemas físicos.

Foram utilizados quatro computadores, além do *switch* D_LINK DES 3526 que possuí uma conexão com a rede *Ethernet* do DCCE (Internet), conforme o cenário abaixo.

Figura 33 – Cenário para Teste de Funcionamento.

O sistema Operacional escolhido foi o Ubuntu GNU/Linux versão 10.04 e 10.10 de código aberto e livre. Esta distribuição é baseada no sistema operacional Debian, escolhido pelo pacote DRAGON para rodar suas máquinas virtuais.

Analisou-se o funcionamento do gerenciamento do canal de controle, funcionalidade mais importante do presente trabalho, já que a correlação dos enlaces TE só serviram para base das funcionalidades opcionais do LMP, a verificação de enlace e a localização de falha.

Para ativar um canal de controle, uma mensagem Config foi transmitida ao nó remoto, e na resposta, recebeu uma mensagem de ConfigAck do nó local. O lmpcc, responsável pelo gerenciamento do canal de controle, foi inicializado trocando as mensagens Config, conforme a Figura 34 abaixo.

Figura 34 – Inicialização do Gerenciamento do Canal de Controle.

A mensagem Config contêm a identificação local do canal de controle (CC_Id), o identificador do nó Node_Id, um Message_Id para mensagem de confiança, além dos parâmetros, conforme mostra a ampliação abaixo.

Figura 35 – Inicialização do Gerenciamento do Canal de Controle.

Os parâmetros são recebidos pelo nó remoto, e então negociados. A Figura 36 mostra o nó remoto recebendo os identificadores e seus parâmetros. O número a frente da **mensagem_rec** indica qual mensagem esta sendo recebida, no caso 5, o qual indica a mensagem Config. Em seguida, recebe os parâmetros de cada identificador, como exemplo Tipo1500 e Tipo500 que representam os parâmetros HelloDeadInterval e HelloInterval, enviados pelo identificador CONFIG.

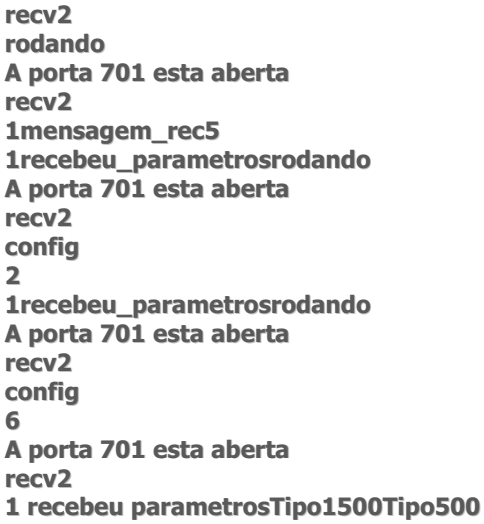


Figura 36 – Recebendo mensagens Config no Gerenciamento do Canal de Controle.

E quando todos os parâmetros foram aceitos, ou seja, o primeiro nó que emitiu a Config enviou a mensagem de ConfigAck, as mensagens Hello puderam ser emitidas e usadas para manter a conectividade do canal e detectar falhas rapidamente (Figura 37).

Figura 37 – Trocas de Mensagens Hello.

Para LMP, é essencial que um canal de controle esteja sempre disponível para um enlace. Desta forma, no caso de uma falha na conexão, o DRAGON busca o reinício automático. A automatização do reinício é a principal característica do gerenciamento do canal de controle, podendo ocorrer por vários fatores como: queda de energia, congestionamento da rede, falha na placa de rede, etc...

A Figura 38 mostra o gerenciamento do canal de controle reiniciando após uma falha mecânica da placa de rede.

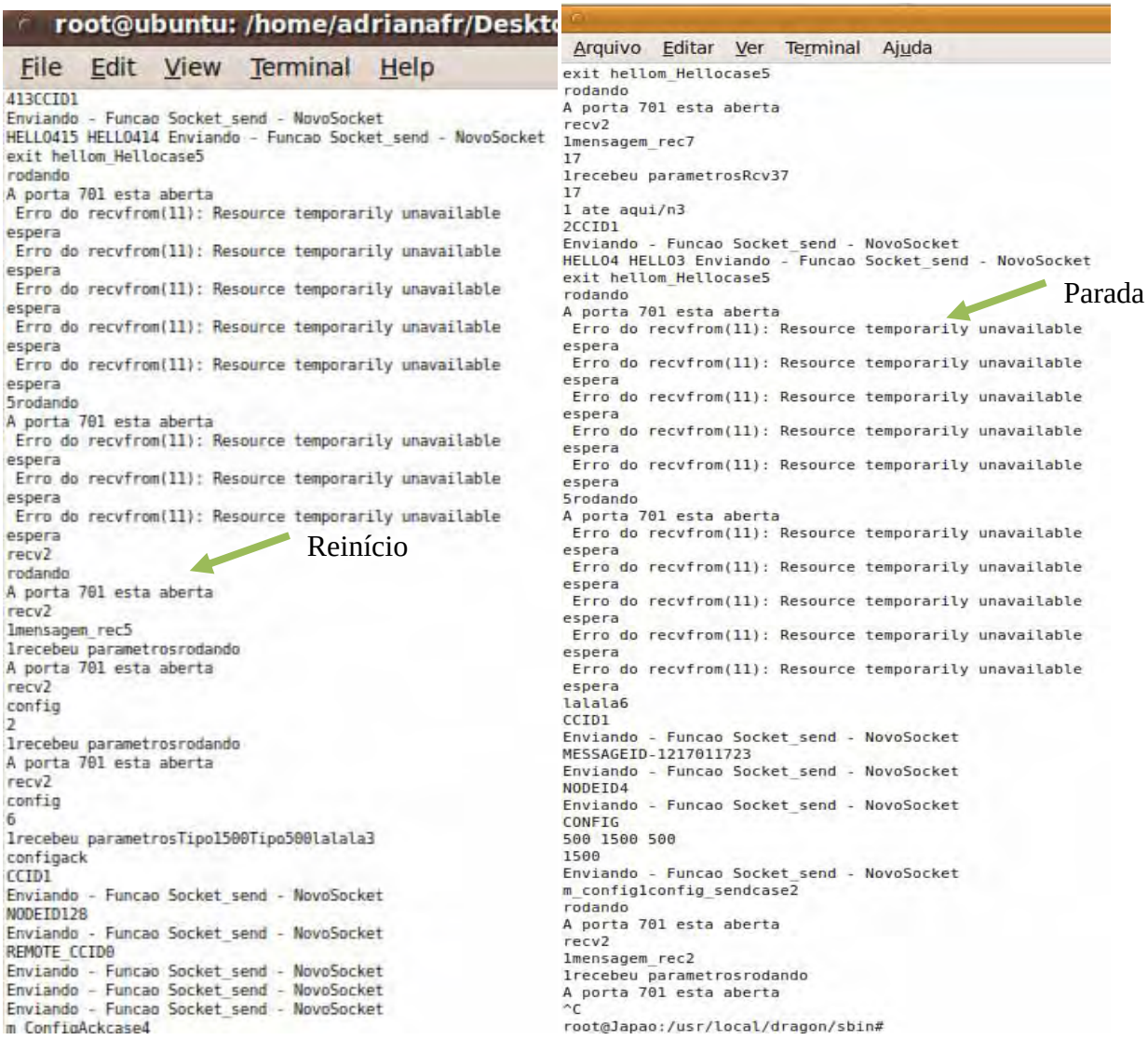


Figura 38 – Reinício Automático após Falha de Rede.

Uma mensagem de erro é enviada pelo socket quando detectada uma flalha na comunicação entre dois nós, conforme mostra a Figura 39.

Figura 39 – Falha de Rede.

Depois que o tempo do HelloDeadInterval se esgota é feita tentativa para reiniciar o canal, se este não detectar mais falha, o canal de controle é então reiniciado, e seus parâmetros negociados novamente.

```
CCID1
Enviando - Funcao Socket_send - NovoSocket
MESSAGEID16728053
Enviando - Funcao Socket_send - NovoSocket
NODEID4
Enviando - Funcao Socket_send - NovoSocket
CONFIG
500 1500 500
1500
Enviando - Funcao Socket_send - NovoSocket
```

Figura 40 – Reinício Automático (Enviando).

```
recv2
rodando
A porta 701 esta aberta
recv2
1mensagem_rec5
1recebeu_parametrosrodando
A porta 701 esta aberta
```

Figura 41 – Reinício Automático (Recebendo).

Entretanto, caso ocorra uma falha do canal de controle e este não consiga reiniciar, o canal de apoio (canal vizinho) deve estar disponível restabelecendo a comunicação com o nó vizinho. Assim, a fim de verificar se o sistema se comportaria desta forma, a Figura 42 mostra o cenário que foi utilizado para este teste.

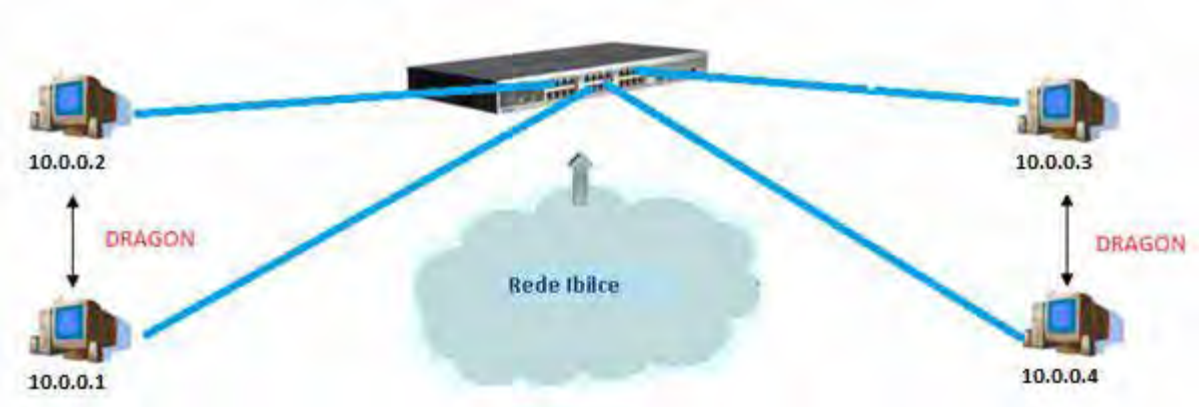


Figura 42 – Cenário Teste de Canal de Apoio.

Para a realização do Teste de Canal de Apoio foram utilizadas quatro máquinas, onde em cada par (10.0.0.1/10.0.0.2 e 10.0.0.3/10.0.0.4) estava habilitado um canal de controle. Porém, quando a máquina 10.0.0.1 parou de funcionar e não restabeleceu conexão, a máquina 10.0.0.3 que estava com o canal de controle funcionando normalmente, iniciou a conexão com a máquina 10.0.0.2, comportando-se como um servidor, restabelecendo a conexão e provando ainda que pode existir mais de um canal de controle em um mesmo nó (Figuras 43 - 51).

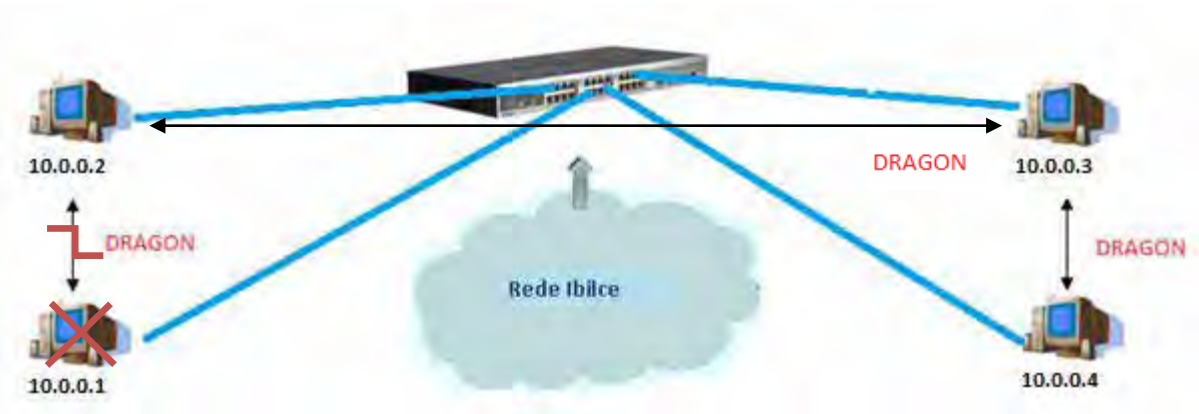


Figura 43 – Cenário Teste de Canal de Apoio após Queda da Máquina.

A Figura 44 mostra o momento em que a máquina 10.0.0.1 foi parada, simulando um falha.

Figura 44 – Máquina 10.0.0.1 - Parada de uma das Máquinas com DRAGON.

A última mensagem Hello recebida foi a de número 203, e a última enviada antes da falha foi a de número 204, conforme a Figura 45.

Figura 45 – Máquina 10.0.0.1 - Momento da Parada.

Neste momento, a máquina que estava com o canal de controle em par com a 10.0.0.1 perde a conexão. Esta, mostrada na Figura abaixo, tentou o reinício.

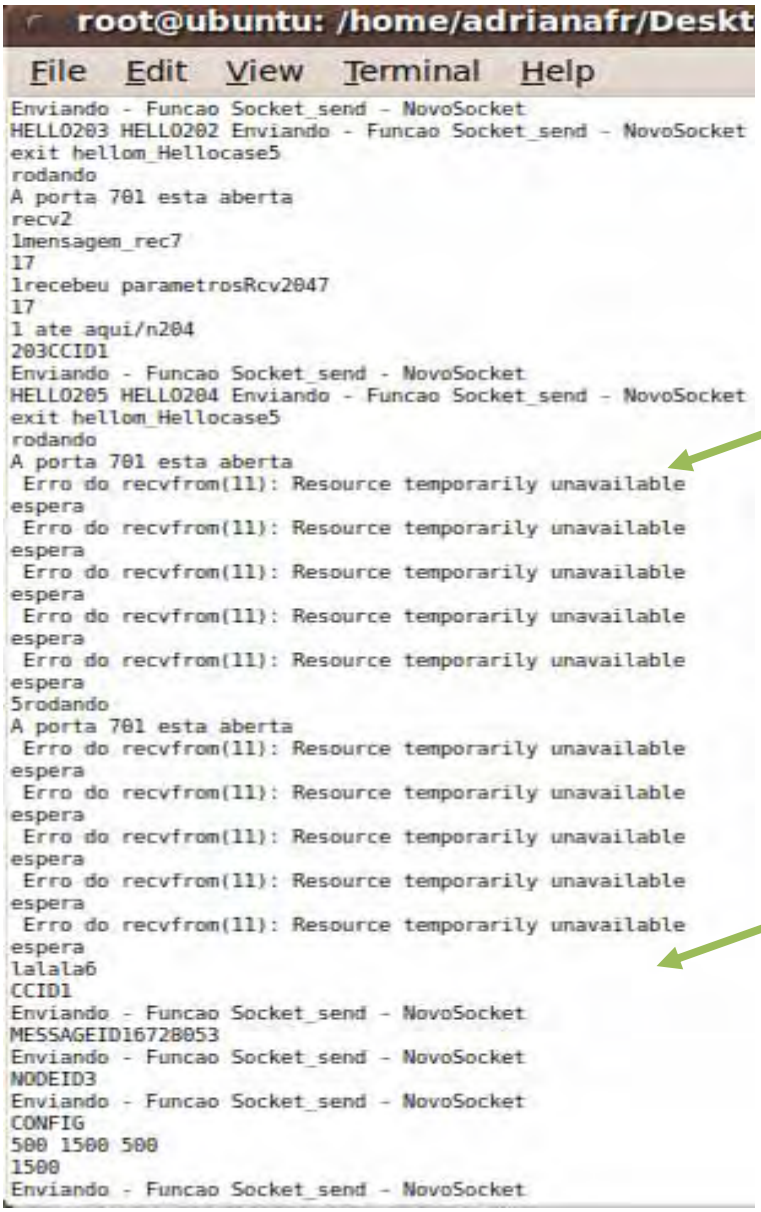


Figura 46 – Máquina 10.0.0.2 - Executando o DRAGON e Reiniciando após Perda de Conexão.

A Figura 47 mostra o momento em que a máquina 10.0.0.2 perde a conexão com a 10.0.0.1, que falhou. Nota-se que a última mensagem Hello enviada pela 10.0.0.1, a de número 204, foi recebida.

```
HELLO205 HELLO204 Enviando - Funcao Socket_send - NovoSocket
Exit hellom_Hellocase5
Rodando
A porta 701 esta aberta
Erro do recvfrom(11): Resource temporarily unavailable
espera
Erro do recvfrom(11): Resource temporarily unavailable
espera
```

Figura 47 – Máquina 10.0.0.2 - Perda de Conexão.

Após o período de espera do HelloDeadInterval e falha na nova tentativa de reconectar o canal de controle, a máquina 10.0.0.2 vai enviar uma mensagem *multicast* para os vizinhos e aguardar que algum nó aceite sua conexão, como apresentado a Figura 48.

```
Erro do recvfrom(11): Resource temporarily unavailable
espera
Erro do recvfrom(11): Resource temporarily unavailable
espera
CCID1
Enviando - Funcao Socket_send - NovoSocket
MESSAGEID16728053
Enviando - Funcao Socket_send - NovoSocket
```

Figura 48 – Máquina 10.0.0.2 - Envio de *multicast* após Perda de Conexão.

Então, a máquina 10.0.0.3 que iniciou um canal de controle com a 10.0.0.4, recebe a mensagem *multicast* enviada, e aceita a conexão solicitada, conforme a Figura 49.

Figura 49 – Máquina 10.0.0.3 - Servidora que Aceitou a Conexão.

A máquina considerada “servidora” terá então dois canais de controle. O primeiro com a 10.0.0.4, que no momento do teste recebia uma mensagem Hello (representada pelo número 7), conforme mostra a Figura 50. Outra com a 10.0.0.2 que havia perdido a conexão.

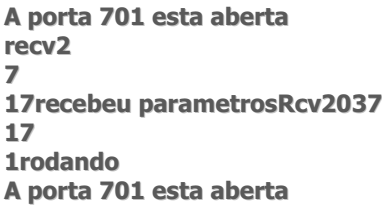


Figura 50 – Máquina 10.0.0.3 - Recebendo da máquina 10.0.0.4.

Observa-se na Figura 51 que mesmo a máquina encontrando uma servidora e estabelecendo novamente a conexão, o reinício é necessário para que os parâmetros dos Config sejam avaliados e aceitos, visto que cada canal de controle tem que haver uma negociação única.

```
Erro do recvfrom(11): Resource temporarily unavailable
espera
Erro do recvfrom(11): Resource temporarily unavailable
espera
CCID1
Enviando - Funcao Socket_send - NovoSocket
MESSAGEID - 1216651275
Enviando - Funcao Socket_send - NovoSocket
```

Figura 51 – Máquina 10.0.0.3 - Reinício com a máquina 10.0.0.2.

Este teste ainda provou, que assim como no DRAGON, o LMP possui canais de controle que são múltiplos, podendo ser ativos simultaneamente entre um par de nós.

4.3. Avaliação do Desenvolvimento, Apresentação e Análise dos Resultados

As principais funcionalidades do canal de controle são manter e reiniciar o canal de controle. Já se sabe que o canal de controle reinicia assim que há uma perda de conexão ou falha da rede. Portanto, com o objetivo de saber o limite em que o DRAGON se mantém trafegando em uma rede, foram realizadas simulações em redes *Ethernet* de 100M*Bits*/s e 10M*Bits*/s que receberam pacotes até o limite da banda (dopagem), enquanto uma ferramenta monitorou o *jitter* até o momento em que o canal de controle reiniciava.

Para que fossem realizados os testes no laboratório, foi utilizada a ferramenta de medição e simulação iperf. O iperf é um software livre, do tipo cliente/servidor desenvolvido pelo *National Laboratory for Apllied Network Research* (NLANR), utilizado na análise de desempenho de banda e cálculo de perda de datagramas na rede. Além de medições este software é capaz de gerar simples carga na rede até saturá-la.

Optou-se pelo uso do protocolo UDP para injetar pacotes pelo iperf, já que no TCP há um mecanismo de controle de congestionamento.

O ambiente de teste apresentou inicialmente três computadores apresentados na Figura 52, além um *switch*. Duas máquinas estavam com o DRAGON funcionando e uma terceira só recebendo pacotes UDP, uma vez que o interesse era congestionar o meio (o *switch*) e não a máquina que estava recebendo os pacotes.

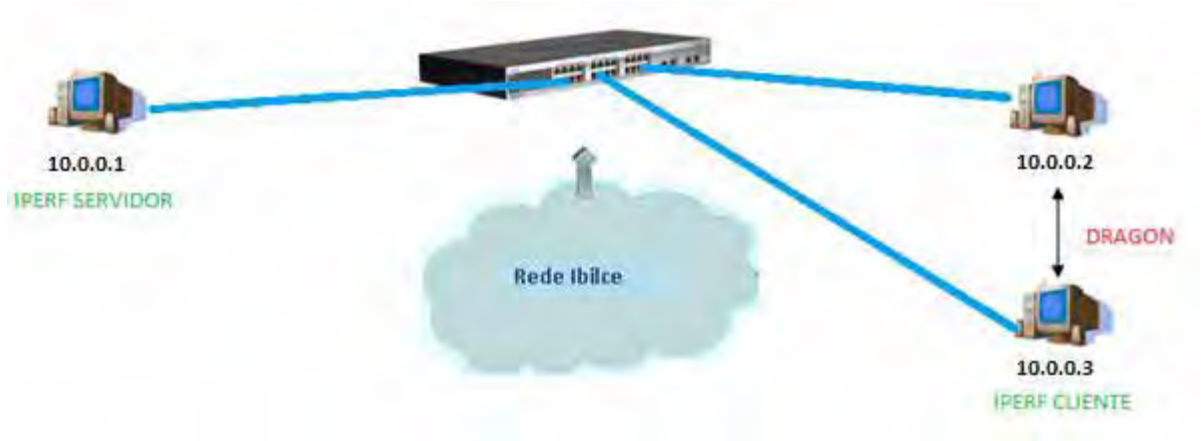


Figura 52 – Cenário1 de Teste com *Switch*.

Neste ensaio a banda utilizada foi de 100*Mbits*/s, mas com o controle de colisão do *switch* a banda não atingiu o limite. Portanto o DRAGON permaneceu enviando mensagens Hello para manutenção do canal no limite que o *switch* permitia (95,7*Mbits*/s) (Figuras 53 e 54).

```
root@China: /home/dragon/jperf-2.0.2
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda

root@China:/home/dragon/jperf-2.0.2# sudo iperf -c 10.0.0.1 -b100M -t6o -u
WARNING: option -b implies udp testing
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:   112 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.2 port 50032 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 6.0 sec  71.9 MBytes    101 Mbits/sec
[  3] Sent 51283 datagrams
[  3] Server Report:
[  3]  0.0- 6.0 sec  71.9 MBytes    101 Mbits/sec  0.000 ms    0/51282 (0%)
[  3]  0.0- 6.0 sec  1 datagrams received out-of-order
root@China:/home/dragon/jperf-2.0.2#
```

Figura 53 – Iperf Cliente.

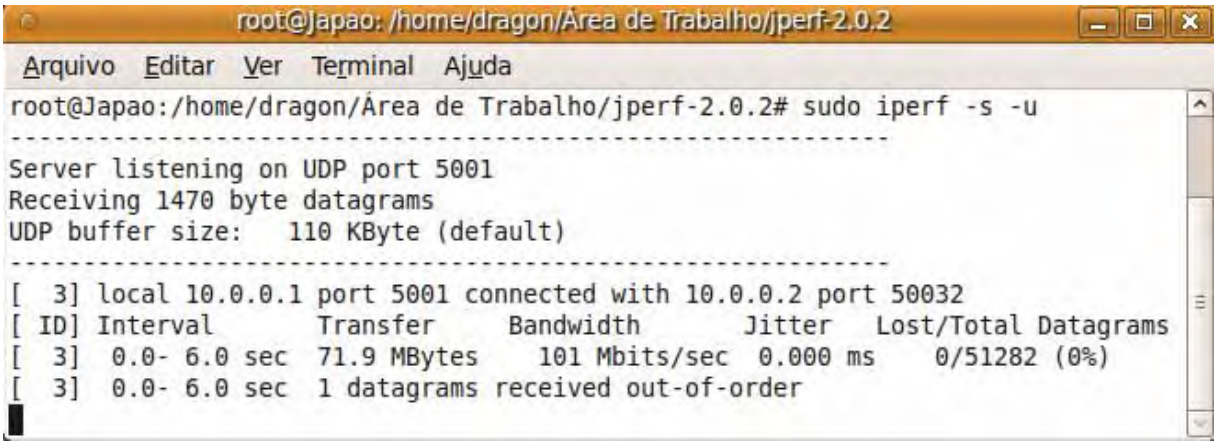


Figura 54 – Iperf Servidor.

Foram realizados 50 testes com o iperf, onde enviou-se 100*MBytes* em 60s, e em nenhum momento o DRAGON reiniciou.

O segundo cenário possuía três máquinas, além do *swicth* e de um *Hub* IBM 8242-008 conectados conforme a Figura 55, onde foi utilizada uma banda menor de 10*Mbits*/s (limitada pelo *Hub*).

Figura 55 – Cenário 2 de Teste com *Hub* e *Switch*.

Como o *Hub* não utiliza controle de colisão, foi possível alcançar o limite de 10*Mbits*/s de banda. Foram então executados 50 testes com as bandas em M*Bits*/s de 8.6, 8.7, 8.8, 8.9, 9.0, 9.1, 9.2, 9.3, 9.4, 9.5, 9.6 9.7, 9.8, 9.9 e 10 , com envio de 10*MBytes* e tempo de espera de 60s. Um outro item analisado foi o atraso (*jitter*) do pacote no momento em que o DRAGON caiu sua conexão. A Figura 56 mostra os 50 testes, o momento que a rede caiu e a quantidade média de atrasos dos pacotes na banda em que o DRAGON reiniciou.

Testes de Desempenho do DRAGON na rede																
Tentativas	Banda 8,6Mbits/s	8,7	8,8	8,9	9	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9	10	Atraso na Perda de Conexão no DRAGON
1	Não	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	3,581ms
2	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	0,114ms
3	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	0,069ms
4	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	1,892ms
5	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	1,531ms
6	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,195ms
7	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	1,797ms
8	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	2,147ms
9	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	0,083ms
10	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,138ms
11	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,211ms
12	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,679ms
13	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,775ms
14	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,558ms
15	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,729ms
16	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	2,334ms
17	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	1,903ms
18	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,248ms
19	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,430ms
20	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	3,455ms
21	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,793ms
22	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	3,137ms
23	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	3,670ms
24	Não	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	2,706ms
25	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	0,765ms
26	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,486ms
27	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,985ms
28	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,941ms
29	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	2,285ms
30	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,793ms
31	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,221ms
32	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,401ms
33	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,048ms
34	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,945ms
35	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,643ms
36	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,502ms
37	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,094ms
38	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,777ms
39	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	2,205ms
40	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,320ms
41	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	0,987ms
42	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,235ms
43	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	0,766ms
44	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	2,207ms
45	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	2,995ms
46	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,583ms
47	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	0,998ms
48	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	1,231ms
49	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	0,187ms
50	Não	Não	Não	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim	Sim	Sim	Sim	0,542ms

Legenda
Sim – Limite da banda em que o DRAGON reiniciou
Não - Limite da banda em que o DRAGON não reiniciou

Figura 56 – Teste de Desenvolvimento com 10*Mbits*/s.

Com base nos resultados apresentados na tabela, observa-se que raras vezes o DRAGON reiniciou antes de 9,5 *Mbits*/s e, a partir de 9,8*Mbits*/s, o DRAGON já não conseguiu se restabelecer enquanto a rede permanecia saturada. Para visualizar melhor seu limite, os reinícios do DRAGON foram abordados de forma quantitativa e gerou-se o gráfico abaixo.

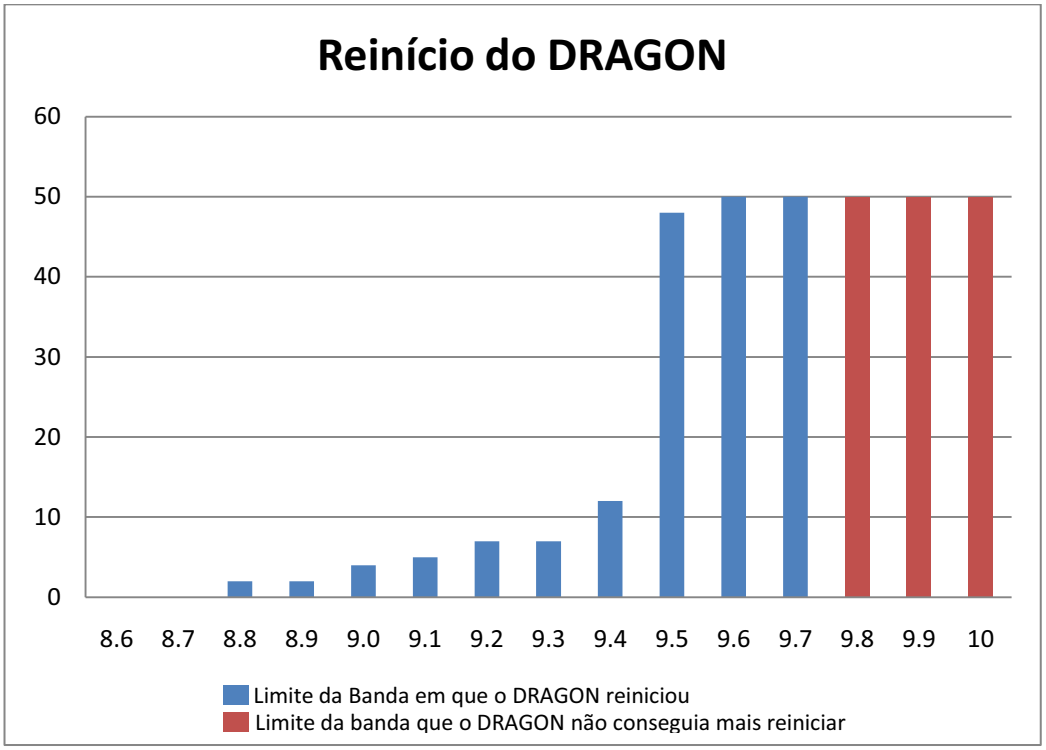


Figura 57 – Quantidade de Reinícios do DRAGON por Banda da Rede.

Pode-se notar no gráfico, que na maioria dos testes, o DRAGON reiniciou em 9,5*MBits*/s. Acima de 9,8*MBits*/s, que se encontra em vermelho no gráfico, o DRAGON não conseguiu mais restabelecer a conexão, enquanto os pacotes UDPs estavam sendo injetados, causando o congestionamento. Assim que o teste foi finalizado (em torno de 60 segundos), o DRAGON novamente tentou a conexão e voltou a comunicar com o nó vizinho.

Outra medida utilizada foi encontrar a média dos atrasos dos pacotes no período em que o DRAGON reinicia. A Figura 58 abaixo mostra os atrasos e também uma linha de tendência onde se encontraria a média dos atrasos.

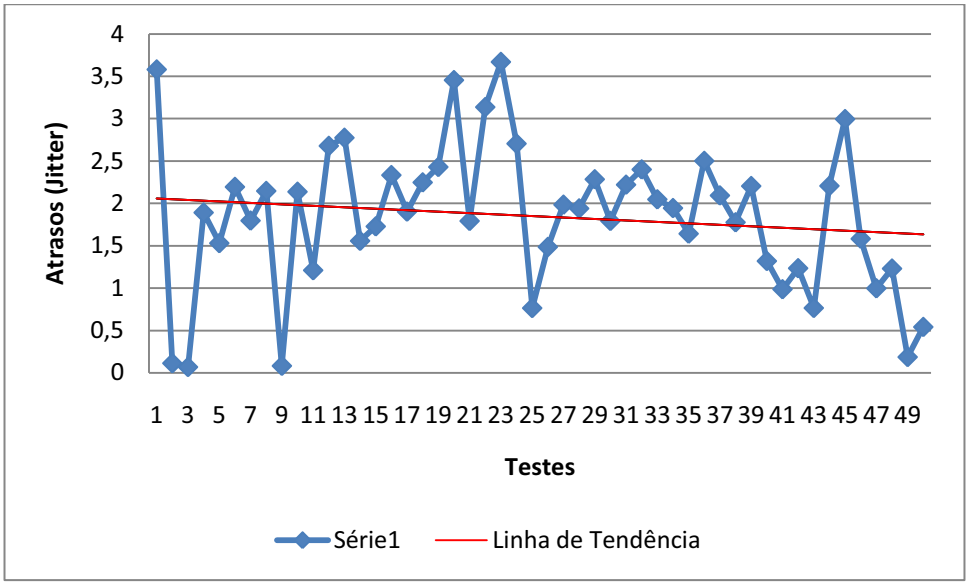


Figura 58 – Média dos Atrasos no Momento do Reinício do DRAGON.

A linha de tendência (em vermelho) varia entre 1,6 a 2ms aproximadamente nos 50 testes realizados. Esse atraso pode ser considerado pequeno, devido período de transmissão do pacote na rede. Portanto conclui-se que a perda de conexão é causada pelas colisões no *hub*, quando se aumenta a banda e satura o enlace.

5. CONCLUSÕES

Apesar de ainda poucos, houve o surgimento de trabalhos relacionados ao protocolo LMP dentro da arquitetura GMPLS, nos últimos anos, principalmente após a RFC 4204 em 2005. Isso se deve ao número pequeno de instituições que possuem redes experimentais de alta velocidade e projetos nessa rede.

Com a proposta de contribuir para uma solução nacional de equipamentos de gerenciamento óptico, o protocolo de gerenciamento de enlace, o LMP, objeto de estudo do presente trabalho, teve duas de suas funcionalidades implementadas em versão *open-source*: o gerenciamento do canal de controle e a correlação das propriedades dos enlaces, conforme a RFC 4204. Após implementado, o LMP foi acrescentado ao DRAGON, devido a este ser um pacote com as ferramentas de gerenciamento distribuído GMPLS, que possui os protocolos de roteamento e sinalização já *open-source*.

A funcionalidade de correlação de propriedade foi implementada, porém não testada, já que ela depende de dois parâmetros que só são inicializados na verificação da conectividade do enlace, que pela RFC 4202 é considerada opcional.

Testes foram realizados e provaram que o DRAGON, com protocolo LMP integrado, conseguiu comprovar com os resultados experimentais obtidos, as funcionalidades do protocolo relacionadas ao estabelecimento, manutenção e gerenciamento do canal de controle, além da identificação das propriedades dos nós adjacentes e isolação de falhas simples ou múltiplas no domínio óptico do gerenciamento do enlace. Além do pacote conseguir se manter trafegando em uma rede com o tráfego no limite da banda.

Desta forma, o protocolo LMP implementado no pacote *open-source* DRAGON poderá ser então disponibilizado e integrado aos equipamentos de diversos tipos (como o ROADM - Multiplexador de Inserção/Remoção Óptico Reconfigurável) e fabricantes em curto prazo, apresentando-se como uma solução nacional com preços mais acessíveis.

5.1. TRABALHOS FUTUROS

Como atividade de continuidade ao presente trabalho, a implementação das outras duas funcionalidades do LMP (verificação da conectividade do enlace e localização de falhas) poderá ser realizada, permitindo que testes de funcionamento possam ser realizados em ambientes de reais de redes de fibra ópticas, assim que disponibilizar o sinal do backbone Kyatera[2] para o laboratório LACE (Laboratório de Automação e Computação Evolutiva) no Departamento de Ciências de Computação e Estatística (DCCE) da UNESP de São José do Rio Preto.

[2]Backbone Kyatera – Iterliga laboratórios das universidades do Estado de São Paulo

REFERÊNCIAS BIBLIOGRÁFICAS

ADEEL, M.; JAVED, S.; CHAUDHRY, A. A.; ZAIDI, M. H.; MAHMOOD, W.; IQBAL, A. A. **A Comparative Analysis of Routing Protocols in GMPL**. NUST Institute of Information Technology e National University of Sciences and Technology Pakistan. *The First IEEE and IFIP International Conference in Central Asia*, 26-29 Setembro de 2005.

ALOIA, E. J. **Contribuições para a Análise e Simulação de Redes Ópticas: Aspectos de Engenharia de Tráfego, Restauração Dinâmica e Conversão de Comprimentos de Onda.** Tese (Doutorado em Engenharia Elétrica), Universidade de São Paulo, SP, Brasil, 2009.

ALOIA, E. J. **Sistematização Crítica das Tendências de Padronização de Arquiteturas e Protocolos em Redes Ópticas.** Dissertação (Mestrado em Engenharia Elétrica) - Escola de Engenharia de São Carlos (EESC), São Carlos, SP, Brasil. 2003.

BENERJEE, A.; DRAKE, J.; LANG, J.; TURNER, B.; **Generalized Multiprotocol Label Switching: An Overview of Signaling Enhancements and Recovery Techniques.** Topics in Internet Technology - IEEE Communications Magazine, Julho de 2001.

BARROS, M. R. X.; ROCHA, M. L.; SIMÕES, F. D.; ROSOLEM, J. B.; ROSSI, S. M.; FUDOLI, T.R.T. **Soluções para redes ópticas avançadas: Projeto GIGA.** Cad. CPqD Tecnologia, Campinas, v. 1, n. 1, p. 37-60, 2005.

BERGER, L. **Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description.** Internet Engineering Task Force. IETF RFC 3471, Janeiro de 2003. *http://www.ietf.org.*

BRASSOLATI, I. R. **Dimensionamento de Rede GMPLS com Base em Algoritmos RWA**. Dissertação (Mestrado em Engenharia de Elétrica) - Escola Politécnica da Universidade de São Paulo, SP, Brasil, 2006.

CHEGE, N. L. & FREELANCE, B. **Performance Evaluation of MPLS/GMPLS Control Plane Signaling Protocols.** Dissertação (Mestrado em Engenharia Elétrtica) - Blekinge Institute of Technology, Karlskrona, Suécia, 2009.

CRISPIM, H. A. F. **Implementação de um Sistema de Controle Centralizado para uma Rede Óptica Transparente**. Tese (Doutorado em Engenharia Elétrica) – Universidade de Brasilía, Brasilía, DF, Brasil, Agosto de 2006.

CUNHA, D. V. **Análise Lógica de Protocolos, Proposta e Avaliação de Desempenho de um Algoritmo de Atribuição de Rótulos Baseado em SRLG em um Ambiente GMPLS-WDM**. Dissertação (Mestrado em Engenharia de Computação) - Escola Politécnica da Universidade de São Paulo, SP, Brasil, 2006.

DRAGON Project Website. 1999-2008. Disponível no site em: < http://dragon.maxgigapop.net >. Acesso em: 25 Agosto 2008.

DRAGON Control Plane Overview, Version 1.0a. 2006. Disponível no site em: <http://dragon.east.isi.edu >. Acesso em: 4 Fevereiro 2008.

DRAKE, J.; KIRETI, K.; YAKOV, R.; DEBANJAN, S.; LOU, BERGER.; DEBASHIS, B. **Link Management Protocol (LMP),** DRAFT IETF - draft-lang-mpls-lmp-00.txt, Setembro 2000. *http://www.ietf.org.*

DUBUC, M.; NADEAU, T.; LANG, J.; MCGINNIS, E. **Link Management Protocol (LMP) Management Information Base (MIB).** Internet Engineering Task Force. IETF RFC 4327, Janeiro de 2006. *http://www.ietf.org.*

ESPINOSA, C.A.L.; ALBAN, D.C.; BERMUDEZ, C.T. **Lectura Automática de Medidores de Consumo de Agua com Tecnología Bluetooth**. *4°Congresso De Computação da Colômbia 4CC* – UNAB –UIS. 2009

FEDYK, D.; BERGER, L.; ANDERSON, L. **Generalized Multiprotocol Label Switching (GMPLS) Ethernet Label Switching Architecture and Framework**. Internet Engineering Task Force. IETF RFC 5828, 2010. *http://www.ietf.org.*

FEDYK, D.; BRUNGARD D.; LANG, J.; PAPADIMITRIOU, D. **A Transport Network View of the Link Management Protocol (LMP).** Internet Engineering Task Force. IETF RFC 4394, Fevereiro de 2006. *http://www.ietf.org.*

FREDETTE, A. & LANG, J. **Link Management Protocol (LMP) for Dense Wavelength Division Multiplexing (DWDM) Optical Line Systems.** Internet Engineering Task Force. IETF RFC 4209, Outubro de 2005. *http://www.ietf.org.*

FREDETTE, A.; SNYDER, E.; SHANTIGRAM, J.; DRAKE, J.; TUMULURI, G.; GOYAL, R.; BORSON, S.; KRISHNAN, R.; EDWARDS, W.L.; XUE, Y.; YU, J. **Link Management Protocol (LMP) for WDM Transmission Systems.** DRAFT IETF - Draft-fredette-lmp-wdm-oo.txt, Dezembro de 2000. *http://www.ietf.org.*

HAYASHI, R. & SHIOMOTO, K. **Link Management Protocol exttensions for optical plug and play.** DRAFT IETF - Draft-hayashi-ccamp-lmp-pnp-ext-00.txt , 5 de Julho de 2010. *http://www.ietf.org.*

HAM, J.V.; VERHOOG, G.J. **Virtual Environments for Networking Experiments**. 1 julho 2004. Disponível no site em: < http://dragon.maxgigapop.net/twiki/pub/DRAGON/UserModeLinux/anp-jvdh-gjv.pdf. >. Acesso em: 20 Maio 2010.

HUANG, S.; LI, B.; DENG, Y.; ZHANG, J.; GU, W. **A GMPLS-based Recovery Mechusing Enhanced LMP**. *The Journal of China Universities of Posts and Telecommunications,*Volume 15, Edição 1, Março de 2008.

KOMPELLA, K. **Routing Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS).** Internet Engineering Task Force. IETF RFC 4202, Outubro de 2005. *http://www.ietf.org.*

JULIÁN, D.G. **Estudio de la integración de los protocolos de GMPLS en redes OBS**. Dissertação (Mestrado em Engenharia Telemática) - Universidade de Catalunha, Departamento de Engenharia Telemática, 12 de fevereiro de 2009.

LANG, J.; MITRA, K.; DRAKE, J.; KOMPLELL, K.; REKHTER, Y.; SAHA, D.; BERGER, L.;BASAK, D.; **Link Management Protocol (LMP).** DRAFT IETF - Draft-lang-mpls-lmp-00.txt, Setembro de 2000. *http://www.ietf.org.*

LANG, J. **Link Management Protocol (LMP)**. Internet Engineering Task Force. IETF RFC 4204, 2005. *http://www.ietf.org.*

LEE, J.H.; YOSHIKANE, N.; TURITANI, T.; OTANI, T. **Optical Link Performance Monitoring using Extended Link Management Protocol for Transparent Optical Networks.** *Conference on Optical Fiber Communication*. OSA/OFC/NFOEC 2009, San Diego – CA, USA, 22-26 Março de 2009

LEHMAN, T.; SOBIESKI, J.; JABBARI, B. **DRAGON: A Framework for Service Provisioning in Heterogeneous Grid Networks.** IEEE Communications Magazine, Vol 44, Issue 3, pg 84-90, 20 de Março de 2006.

LOUREIRO, A.A.F.; NOGUEIRA, J.M.S.; RUIZ, L.B.; MINI, R.A.F.; NAKAMURA, E.F.; FIGUEIREDO, C.M.S. **Redes de Sensores Sem Fio**. *XXI Simpósio Brasileiro de Redes de Computadores*, Natal, CE, Brasil, 2003.

LUO, T. & KUO, G.S. **Analyses and Improvements of Link Management Protocol For GMPLS-based Networks**. *IEEE Global Telecommunications Conference - GLOBECOM 2003,*Vol 5 pg 2992-2998, 1-5 Dezembro 2003.

MANNIE, E. **Generalized Multi-Protocol Label Switching (GMPLS) Architecture.** Internet Engineering Task Force. IETF RFC 3945, 2004. *http://www.ietf.org.*

PAPADIMITRIU, D.; BERDE, B.; MARTINEZ, R.; ORDAS, J.G.; THEILLAUD, R.; VERBRUGGE, S. **Generalized Multi-Protocol Label Switching (GMPLS) Unified Control Plane Validation**. *IEEE International Conference on Communications, 2006*, Vol 6 pg 2717-2714, Junho de 2006.

PASQUINI, R. **Arquitetura e Prototipação de uma UNI Óptica para a Rede GIGA**. Dissertação (Mestrado em Engenharia Elétrica, Área de concentração: Engenharia de Computação) - Universidade Estadual de Campinas - Faculdade de Engenharia Elétrica e de Computação, Campinas, SP, Brasil, 2006.

PERELLÓ, J.; SPADARO, S.; COMELLAS, J.; JUNYENT G. **An Analytical Study of Control Plane Failures Impact on GMPLS Ring Optical Networks.** *IEEE COMMUNICATIONS LETTERS*, VOL. 11, NO. 8, Agosto de 2007.

PERELLÓ, J.; SPADARO, S.; COMELLAS, J.; JUNYENT G. **Burst Contention Avoidance Schemes in Hybrid GMPLS-enabled OBS/OCS Optical Networks.** *International Conferende on Optical Network Design and Modeling*, 2009 – ONDM 2009, pg 1-6, Braunscgweig, 18-20 de Fevereiro de 2009.

PERELLÓ, J.; ESCALONA, E.; SPADARO, S.; COMELLAS, J.; JUNYENT G. **Link Management Protocol Automatic Control Plane Configuration Extensions for Resilient Ring-based Arquitectures.** *Conference International on Transparent Opticao Networks*, Vol 3., pg 165-168, Nottingham, 18-22 Junho de 2006.

PERELLÓ, J.; ESCALONA, E.; SPADARO, S.; COMELLAS, J.; JUNYENT G.. **Resource Discovery in ASON/GMPLS Transport Networks.** *Advances in Control And Management of Connection-Oriented Networks.* Universitat Politècnica de Catalunya (UPC). IEEE Communications Magazine, Outubro de 2007.

RUBUYAT, A. R. **Path Computation Element in GMPLS Enable Multi-layer Networks.** Dissertação (Mestrado em Engenharia Elétrica) – KTH Royal Institute of Technology, Estocolmo, Suécia, 2006.

SENO, S.; BABA, Y.; YOSHIDA, S.; KAMEI, M.; HORIUCHI, E.; ONOHARA, T.; IDEGUCHI, T. **Design of a Chromatic Dispersion Measuremente Control Protocol Based upon the GMPLS Architecture**. *Conference on Optical Fiber Communication OFC 2009,* San Diego, CA, 22 – 26 de Março de 2009.

SHAN-GUI, H.; BIN, L.; YU, D.; JIE, Z.; WAN-YI, G. **Syatematic and Efficient GMPLS-bsaed recovery mechanism with soft-protection using enhanced LMP**. *The Journal of China Universities of Posts and Telecomunications,* Vol 15, Issue 1, Março de 2008.

SHIMIZU, K.; INOUE, I.; SHIOMOTO, K. **A Control Channel Management Method for Creating a Reliable and Stable Control Plane for GMPLS-based Optical Transport Networks.** 7th *Asia-Pacific Siymposium on Information and Telecommunication Technologies – APSITT*, Bandos Island, pg 144-148, 22-24 de Abril de 2008.

SHIMIZU, K.; HAYASHI, R.; INOUE, I.; SHIOMOTO, K.; **Plug and Play Techniques for GMPLS Control Plane Configuration.** *7th International Conference on Optical Internet* – COIN 2008. Tokyo, 14-16 de Outubro de 2008.

SILVA, E. R. T. **Internet Baseada em Redes Ópticas**. Dissertação (Mestrado Profissional em Computação) – Universidade Estadual de Campinas, Campinas, SP, Brasil, 20 de Dezembro de 2004.

SOUZA, J.V.; ESTEVES, R.P.; ABELÉM, A.J.G.; COSTA, J.C.W.A; STANTON, M.A.; SANTOS FILHO, O.G. **Análise de Mecanismos de Recuperação de Falhas em Redes Ópticas de Nova Geração com Plano de Controle Baseado no GMPLS**.*XXV Congresso as Sociedade Brasileira de Computação* , UNISIMOD – São Leopoldo/RS, 22-29 de Julho de 2005.

TAGAME, A.; HASEGAWA, T. **A Study of IP Routing Stability for GMPLS-Based Next-Generation Backbone Network.** *Electronics and Communcations in Japan,* Part 1, Vol 90 No. 11, 2007, Translated from Denshi Joho Tsushin Gakkai Ronbunshi, Vol J89-B, No. 2, pg 88-96, Fevereiro 2006.

TING, W.; JUN-DA, H.; CHENJIE, G.; JIA, Z.; WEI, Y. **Wireless Monitoring System Based on Bluetooth smart phones**. *2010 International Conference on Networking and Digital Society*, pag 648-651, Wenzhou, 30-31 Maio de 2010.

TSURITANI T. & OTANI T. **Demonstration of Interworking Operation between DWDM and PXC for Reliable All-Optical Switching Networks.** T*he 20th Annual Meeting of the IEEE Lasers and Electro-Optics Society – LEOS 2007*, pg 743-744, Lake Buena Vista, FL, 21-25 de Outubro de 2007.

TOMBI, R. G. **Proposta de Extensão do Protocolo PCE Considerando Parâmetros de QoT para Roteamento Inter-Domínios em Redes GMPLS**. Dissertação (Mestrado em Engenharia de Computação) - Escola Politécnica da Universidade de São Paulo, São Paulo, SP, Brasil, 2007.

VELASCO, L.; SPADARO, S.; COMELLAS, J,; JUNYENT, G.**ROADM design for OMS-DPRing protection in GMPLS-based optical networks**. *6th International Workshop on Design and Reliable Communication Networks – DRCN 2007*, La Rochelle, 7-10 de Outubro de 2007.

VELASCO, L.; SPADARO, S.; COMELLAS, J,; JUNYENT, G. **Experimental Evaluation of OMS Protection in GMPLS-Based Optical Networks**. *Optical Communications Group, Signal Theory and Communications*. Department Universitat Politecnica de Catalunya (UPC), C Jordi Girona, 1-3, Barcelona, Espanha, 2007.

VERDI, F. L. **Uma Arquitetura para Provisionamento e Gerência de Serviços em Redes Ópticas**. Tese (Doutorado em Engenharia Elétrica, Área de concentração: Engenharia de Computação) - Universidade Estadual de Campinas - Faculdade de Engenharia Elétrica e de Computação, Campinas, SP, Brasil, 2006.

YE, J.Y. Atlantis: **Location Based Services with Bluetooth**. Department of Computer Science Brown University. Undergraduate Honors Theses. Maio de 2005. Disponível em: < http://list.cs.brown.edu/research/pubs/theses/ugrad/>. Acesso no site em: 12 Maio 2009.

ZANGH, N. & BAO, H. **Research on Application of GMPLS Network in Core Optical Network**. *Second International Workshop on Education Technology and Computer Science (ETCS2010)*, Wuhan, China, 6-7 Março de 2010.

APÊNDICE A – Códigos em C

MEF do Canal de Controle

Dentro da função gerenciamento_canal(), a operação do canal de controle de LMP é descrita nos termos da MEF em estados e eventos. Cada evento tem seu número e nome simbólico. Estes eventos foram implementados em linguagem C e apresentados abaixo:

1: **evBringUp:**

1a) a emissão de uma mensagem Config,

```
ret_send = m_Config(p);
        if (ret_send ==1){ // Se enviou corretamente retorna 1;
            ESTADO = 2;
break;
    }
```

1b) um período da espera para receber uma mensagem Config do nó remoto.

```
while(1){
sockc =funcao_socket_recv(p);
    if(sockc != 5) { //Se recebu alguma mensagem diferente de erro
        if (mensagem_rec(p)== 1) { // Verifica se a mensagem recebida é 1=Config
                ESTADO = 3;
                break;
        break;
    }
}
```

2: **evCCDn**:

```
close (sock); close(socke);
HELLO_INTERVAL = 500;
HELLO_DEAD_INTERVAL =1500;
TxSeqNum= 1;RcvSeqNum=0;
gerenciamento_canal(701);   //volta para o ESTADO = 1;
break;  }break;
```

3: **evConfDone:**

```
if (sockc == 2) { //Recebeu uma mensagem ConfigACK = 2
        ret_send = m_ConfigAck(p);
         ESTADO = 4;
        contador = 0;
        relogio = 0;
        break;
      }
```

4: **evConfErr:**

```
if (sockc== 3) {//Recebeu uma mensagem ConfigNACK = 3
       neg =negociacao_parametros();
       if (neg == ok) { //14 reconfigura
                ret_send = m_Config(parametros);// reevia com os novos parâmetros
           ESTADO = 2;
           break;   }
       else {
        ret_send = m_ConfigNack(parametros,p);
         ESTADO = 2;
         break;
       }
         contador=0;
       relogio=0;
       break;
}
```

5: **evNewConfOK**:

```
rp= reconhece_parametros();
       if (rp==1) {
                ret_send = m_ConfigAck(p);
                printf("Envia Config Ack\n");
                ESTADO= 5;
                contador=0;
              relogio=0;
               break;
          } // fecha if rp== 1
```

6: **evNewConfErr:**

```
else if (reconhece_parametros() ==0) // se parametros rejeitados
        {
        ret_send = m_ConfigNack(p); // Envia uma ConfigNACK
        ESTADO= 1;
        break; }
```

7: **evContenWin:**

```
if (tempo_recebida == tempo) {
   //nó local ganha disputa
        result1 = NODEID; // recebida
        result2 = NODEIDRCV;
         if (result1 < result2) {
        //Ignora Config recebida;
                ESTADO = 2;
       break;}
}
```

8: **evContenLost:**

8a) A mensagem Config é positivamente reconhecida.

8b) A mensagem Config é negativamente reconhecida.

```
if (result1 > result2) {
      rp= reconhece_parametros();
        if (rp==1) {
                printf("enviar ack\n");// se ok
                ret_send = m_ConfigAck(p);
                printf("Envia Config Ack\n");
                ESTADO= 5;
                contador=0;
                relogio=0;
                break;
                } // fecha if rp== 1
        else {  //NACK
                ESTADO = 3;
                contador=0;
```

```
relogio=0;
break;
 }}// fecha
```

9: **evAdminDown:**

```
if (strcmp(lmp.flag, "0x01")==0) //Ativa o Flag que indica para o canal ir para Down
    {ESTADO= 6;
    break; }
```

10: **evNbrGoesDn**:

```
else if (strcmp(lmprcv.flag, "0x01")==0) { //Recebeu um pacote com o Flag que indica que o canal foi para Down
    //10 - Um pacote com o flag ControlChannelDown Ã© recebido do vizinho.
        for(cont=0;cont<550;cont++){}
        close (sock); close(socke);
        HELLO_INTERVAL = 500;
        HELLO_DEAD_INTERVAL =1500;
        TxSeqNum= 1;RcvSeqNum=0;
        gerenciamento_canal(701);       //volta para o ESTADO = 1;
        break;
    }
```

11: **evHelloRcvd:**

```
 if (sockc==4) {  // Recebe uma mensagem Hello = 4
        TxSeqNum=TxSeqNum+1;
        if (RcvSeqNum ==(TxSeqNum)) {
        for    (HELLO_INTERVAL=1;    HELLO_INTERVAL!=0;HELLO_INTERVAL--){}

            ret_send = m_Hello(p);
            //11- Um pacote Hello com o esperado SeqNum foi  recebido.
            ESTADO =5;
        break;
        }
    }
```

12: **evHoldTimer**:

12a) a emissão de mensagens Config periódicas,

12b) um período da espera para receber mensagens Config do nó remoto.

```
if (HelloDeadInterval == 0) {
    //12b  while(contador!=0) {
            if (mensagem_rec() == 2)  //Se recebeu uma mensagem Config
                {...}
    //12a  else ret_send = m_Config(p);  //1ª restransmitido
            relogio--;
            break;
    }
```

13: **evSeqNumErr:** Hello com o inesperado SeqNum recebido e rejeitado.

```
if (RcvSeqNum != TxSeqNum+1){ESTADO=5;}
            HelloDeadInterval --;
             HelloInterval --;
}
```

14: **evReconfig:** Parâmetros do canal de controle foram reconfigurados e requerem a renegociação.

```
if (neg == ok) //14 reconfigura
    {  ret_send = m_Config(parametros);
        ESTADO = 3;
        sockc =mensagem_rec(p);
        break;
        }
```

15: **evConfRet:**

```
//12b  while(contador!=0) {…}
        relogio--;
        break;
        }
 ret_send = m_Config(p);
```

16: **evHelloRet:**

```
if (HelloInterval ==0)
        {ret_send = m_Hello(p);
         ESTADO=5;
         break;
}
```

17: **evDownTimer:**

```
while (contador!=0)
    {
    if (strcmp(lmp.flag,"0x01")== 0)
        ESTADO = 6;
        break;
    }
    contador--;
}
 ESTADO = 6;
```

MEF do Enlace TE

Dentro da função enlace_TE(), a operação de correlação das propriedades dos enlaces de LMP é descrita nos termos da MEF em estados e eventos. Cada evento tem seu número e nome simbólico, assim como na MEF do canal de controle. Estes eventos foram implementados em linguagem C e apresentados abaixo:

1: **evDCUp:**

```
if (ESTADO == 4) ESTADO2 = 3;
    break;
```

2: **evSumAck:**

```
if (sockc  == 6) { //Recebe linkSummary
        neg = reconhece_parametros_TE(); //3 - Não Reconhece Paramêtros
        if (neg == 1){
                ret_send = m_LinkSummaryAck(p);
                ESTADO2 = 3;
                contador = 0;
                relogio = 0;
                break;}
}
```

3: **evSumNack:**

```
if (sockc  == 6) { //Recebe linkSummary
        neg = reconhece_parametros_TE(); //3 - Não Reconhece Paramêtros
        if (neg == 1){…}
        else{ret_send = m_LinkSummaryNack(p);
                ESTADO2 = 2;
                contador = 0;
                relogio = 0;
                break;
}
```

4: **evRcvAck:**

```
if (sockc == 7) {   //7-Este evento indica que uma mensagem de LinkSummaryAck foi recebida, reconhecendo os parâmetros dos Config.
   ret_send = m_LinkSummaryAck(p);
  contador = 0;
  relogio = 0;
    i = i + 1;
    EnlaceTE[i] = INTERFACEID;
    enlace = i;
    printf("%d", enlace);
    ESTADO2 = 3;
    break;
}
```

5: **evRcvNack:**

```
if (sockc == 8) {
        //8 - Quando a mensagem recebida for LinkSummaryNACK
        neg = negociacao_parametros();
        //14 - Parâmetros do canal de controle foram reconfigurados e requerem a renegociação.
        if (neg == ok) { //14 reconfigura
                ret_send = m_LinkSummary(p);
         }
        ret_send = m_LinkSummary(p); //com novos parametros;
        ESTADO2= 3;
        contador=0;
        relogio=0;
        break;
 }
```

6: **evSumRet:**

```
contador = contador * (1 + delta);
relogio=100;
while (relogio!=0) {
        while (contador!=0) {
                if (relógio ==0){
                        ret_send = m_LinkSummary(p);
                else
                        relogio--;
                }
        }
}
```

7: **evCCUp:**

```
if (ESTADO == 5) ESTADO2= 3;
```

8: **evCCDown:**

```
else if(ESTADO ==1) ESTADO2=2;
```

9: **evDCDown:**

```
else ESTADO2=1;
```

A mensagens trocadas nos estados das máquinas de estado finito (MEFs) tanto no gerenciamento do canal de controle, quanto na correlação das propriedades de enlaces TE, são detalhadas à seguir:

Mensagem de Config (tipo dos Msg = 1)

```
int m_Config(int a) { //long int destino
        int ret_send,rs1=0, rs2=0, rs3=0;
        lmp.msg_classe = 1; //1 = Classe Config
        CCID(1,a);
        sleep(1);
        rs1=MESSAGE_ID(1, a);
        sleep(1);
        rs2=NODE_ID(1,a);
        sleep(1);
        rs3=CONFIG(1,a);
        sleep(1);
        printf("m_config");
        if((rs1==5)||(rs2==5)||(rs3==5)){ret_send =5;}
        else{
                ret_send=1;}
         printf("%d", ret_send);
        printf("config_send");
        return ret_send;}
```

Mensagem de ConfigAck (tipo dos Msg = 2)

```
int m_ConfigAck(int a) {
    int ret_send, rs1=0, rs2=0, rs3=0, rs4=0, rs5=0;
    lmp.msg_classe = 2;//2 = Classe ConfigAck
    lmp.obj.classe=1;
    lmp.obj.c_type =1;
    lmp.obj.n = 0;
    lmp.obj.objeto[1] = CCIDV;
    printf("CCID%ld\n",lmp.obj.objeto[1]);
    funcao_socket_send(a);rs1=1;
    sleep(1);
    rs2 = NODE_ID(1,a);
    sleep(1);
```

```
rs3 = CCID(2,a);
sleep(1);
rs4 = MESSAGE_ID(2,a);
sleep(1);
rs5 = NODE_ID(2,a);
sleep(1);
if((rs1==5)||(rs2==5)||(rs3==5)||(rs4==5)||(rs5==5)){ret_send =5;}
        else{
                ret_send=1;}
printf("m_ConfigAck");
return ret_send;}
```

Mensagem de ConfigNack (tipo dos Msg = 3)

```
int m_ConfigNack(int parametros, int a) {
    int ret_send,rs1=0, rs2=0, rs3=0, rs4=0, rs5=0, rs6=0;
    lmp.msg_classe = 3; //3 = Classe ConfigNack
    lmp.obj.classe=1;
    lmp.obj.c_type =1;
    lmp.obj.n = 0;
    lmp.obj.objeto[1] = CCIDV;
    printf("CCID%ld\n",lmp.obj.objeto[1]);
    funcao_socket_send(a);rs1=1;
    sleep(1);
    rs2 = NODE_ID(1,a);
    sleep(1);
    rs3 = CCID(2,a);
    sleep(1);
    rs4 = MESSAGE_ID(2,a);
    sleep(1);
    rs5 = NODE_ID(2,a);
    sleep(1);
    rs6 =CONFIG(1,a);
    sleep(1);
    if((rs1==5)||(rs2==5)||(rs3==5)||(rs4==5)||(rs5==5)||(rs6==5)){ret_send =5;}
    else{
            ret_send=1;}
    printf("m_ConfigNack");
    return ret_send;    }
```

Hello mensagem (tipo dos Msg = 4)

```
int m_Hello(int a) {
    int ret_send, rs1=0, rs2=0;
    lmp.msg_classe= 4; //4 = Classe Hello
    lmp.obj.classe=1;
    lmp.obj.c_type =1;
    lmp.obj.n = 0;
    lmp.obj.objeto[1] = CCIDV;
    printf("CCID%ld\n",lmp.obj.objeto[1]);
    funcao_socket_send(a);rs1=1;
    sleep(1);
    rs2 =  Hello(1,a);
    sleep(1);
    if((rs1==5)||(rs2==5)){ret_send =5;}
    else{
        ret_send=1;}
    printf("m_Hello");
    return ret_send;}
```

Mensagem de LinkSummary (tipo dos Msg = 14)

```
int m_LinkSummary(int a) {
    int ret_send, ct1, ct2, rs1=0, rs2=0, rs3=0;
    ct1 = ctype(11,0);
    ct2 = ctype(12,0);
    lmp.msg_classe = 14; //14 = classe LinkSummary
    rs1 = MESSAGE_ID(1, a);
    sleep(1);
    rs2 = TE_LINK(1, a);
    sleep(1);
    rs3 = DATA_LINK(1, 2, a);// tipo wavelength
    if((rs1==5)||(rs2==5)||(rs3==5)){ret_send =5;}
    else{
        ret_send=1;}
    printf("m_LinkSummary");
    return ret_send;}
```

Mensagem de LinkSummaryAck (tipo dos Msg = 15)

```
int m_LinkSummaryAck(int a) {
     int ret_send, rs1=0;
     lmp.msg_classe = 15; //15 = Classe LinkSummaryAck
    rs1 = MESSAGE_ID(1, a);
    if(rs1==5){ret_send =5;}
    else{
          ret_send=1;}
    printf("m_LinkSummaryAck");
    return ret_send;}
```

Mensagem de LinkSummaryNack (tipo dos Msg = 16)

```
int m_LinkSummaryNack(int a) {
    int ret_send, rs1=0;
    lmp.msg_classe = 16;// 16 = Classe LinkSummaryNack
    rs1 =  MESSAGE_ID(1, a);
    if(rs1==5){ret_send =5;}
    else{
          ret_send=1;}
    printf("m_LinkSumaryNack");
    return ret_send;}
```

Já as definições dos objetos de LMP, que são passadas pelas mensagens acima, possuem o seguinte código:

Classe CCID

```
int CCID(int c_type, int porta){ // Identificador do canal de controle
    long int REMOTE_CCID, CCIDV2=1;
    lmp.obj.classe = 1;
    switch(c_type){
    case 1:   CCIDV= CCIDV2;
          lmp.obj.c_type =c_type;
          lmp.obj.n = 0;
          lmp.obj.objeto[1] = CCIDV;
          printf("CCID%ld\n",lmp.obj.objeto[1]);
```

```
            funcao_socket_send(porta);
            CCIDV2++;
            return 1;
            break;
        case 2:   lmp.obj.c_type =c_type;
            REMOTE_CCID = CCIDRCV;
            lmprcv.objrcv.objeto[1] =REMOTE_CCID;
            printf("REMOTE_CCID%ld\n",REMOTE_CCID);
            lmprcv.objrcv.n = 0;
            funcao_socket_send(porta);
            return 1;
            break;
        default:
        ERROR_CODE(1, "0x10");
        return 5;  }}
```

Classe NODE_ID

```
int NODE_ID(int c_type, int porta){
    long int REMOTE_NODE_ID;
    lmp.obj.classe = 2;
    NODEID ++;
    switch (c_type){
    case 1:lmp.obj.c_type =c_type;
            lmp.obj.objeto[1] = NODEID;
            printf("NODEID%ld\n",lmp.obj.objeto[1]);
            funcao_socket_send(porta);
            lmp.obj.n = 0;
            return 1;
            break;
    case 2:lmp.obj.c_type =c_type;
            REMOTE_NODE_ID =NODEIDRCV;//recebe node id;
            lmprcv.objrcv.objeto[1]= REMOTE_NODE_ID;
            lmprcv.objrcv.n = 0;
            funcao_socket_send(porta);
            return 1;
            break;
    default:
    ERROR_CODE(1, "0x10");
    return 5;       }  }
```

Classe MESSAGE_ID

```
int MESSAGE_ID(int c_type, int porta) {
    long int MESSAGEID, MESSAGEIDRCV;
    lmp.obj.classe = 5;
    MESSAGEID ++;
    MESSAGEIDRCV= MESSAGEID + 1;
    if (c_type==1){
        lmp.obj.c_type =c_type;
        lmp.obj.objeto[1]= MESSAGEID;
        lmp.obj.n = 0;//não negociáveis
        funcao_socket_send(porta);return 1;    }
    else if (c_type==2){
        lmp.obj.c_type =c_type;
        lmp.obj.objeto[1]= MESSAGEIDRCV;
        lmp.obj.n = 0;
        funcao_socket_send(porta);return 1;    }
    else ERROR_CODE(1, "0x10");
    return 5;    }
```

Classe CONFIG

```
    int CONFIG(int c_type, int porta) {
        int i;
        long int  HelloConfig[2];
        printf("CONFIG\n");
        for(i=0;i>=2;i++){
        HelloConfig[i]='0';}
        lmp.obj.classe = 6;
        if (c_type == 1) {lmp.obj.c_type =c_type;
            HelloConfig[0]= HELLO_INTERVAL;
            printf("%ld ",HelloConfig[0]);
            HelloConfig[1] = HELLO_DEAD_INTERVAL;
            printf("%ld ",HelloConfig[1]);
            lmp.obj.n = 1;
            for(i=0;i<2;i++){
                lmp.obj.objeto[i]=HelloConfig[i];
                printf("%ld \n",lmp.obj.objeto[i]);      }
            funcao_socket_send(porta);return 1;    }
        else ERROR_CODE(1, "0x10");return 5;        }
```

Classe HELLO

```
int Hello(int c_type, int porta) {
    long int HELLO_G[2];
    int i;
    lmp.obj.classe = 7;
    memset(&HELLO_G,0,sizeof(HELLO_G));
    for(i=0;i>=2;i++){
    HELLO_G[i]='0';}
    if (c_type == 1) {
        lmp.obj.c_type =c_type;
        if (RcvSeqNum != 0) {
            TxSeqNum = TxSeqNum + 1;
            HELLO_G[0] = TxSeqNum;
            HELLO_G[1] = RcvSeqNum;
            if (TxSeqNum ==0){ ESTADO = 0;
            }
            if (HELLO_G[0] == HELLO_G[1]) {
                strcpy(lmp.flag, "0");
                TxSeqNum= 1;
                RcvSeqNum=0;
                HELLO_G[0] = TxSeqNum;
                HELLO_G[1] = RcvSeqNum;
                ESTADO = 0;    }
            if (HELLO_G[0] == (4294967295)){//((2^32)-1)) {
                TxSeqNum =2;
                HELLO_G[0]=TxSeqNum;   }  }
        else {
            TxSeqNum= 1;//Calculo inicial;
            RcvSeqNum=0;   // Indica que nenhum Hello foi recebido
            HELLO_G[0] = TxSeqNum;
            HELLO_G[1] = RcvSeqNum;
            lmp.obj.n = 0;      }
        for(i=0;i<2;i++){
            lmp.obj.objeto[i]=HELLO_G[i];
            printf("HELLO%ld ",lmp.obj.objeto[i]);      }
        funcao_socket_send(porta);
        printf("exit hello"); return 1;       }
        else ERROR_CODE(1, "0x10"); return 5;}
```

Classe TE_LINK

```
int TE_LINK(int c_type, int porta) {
    char tipo[10];
    struct {
        char flag_tl[4];
        long double Local_Link_Id;
        long double Remote_Link_Id;
        int reserved_tl;
        } te;
    long int IPV4_TE_LINK;
    long double IPV6_TE_LINK;
    long int TE_LINK_UNNUMBERED;
    lmp.obj.classe = 11;
    te.Local_Link_Id ++;
    te.Remote_Link_Id ++;
    strcpy(te.flag_tl,"0x00");/* nao gerência de falha suportada.
    else if (strcmp(te.flag_tl,"0x02")==0) {/*nao suporta verificacao de enlace*/
    //else ERROR_CODE(2, "0x04");
    switch (c_type) {
        case 1:lmp.obj.c_type =c_type;
            IPV4_TE_LINK = te.Local_Link_Id;
            lmp.obj.objeto[1] =IPV4_TE_LINK;
            funcao_socket_send(porta);return 1;
            break;
        case 2:lmp.obj.c_type =c_type;
            IPV6_TE_LINK = te.Local_Link_Id;
            lmp.obj.objeto[1] =IPV6_TE_LINK;
            funcao_socket_send(porta);return 1;
            break;
        case 3:lmp.obj.c_type =c_type;
            TE_LINK_UNNUMBERED = te.Local_Link_Id;
            lmp.obj.objeto[1] = TE_LINK_UNNUMBERED;
            funcao_socket_send(porta);return 1;
            break;
        default:{ ERROR_CODE(1, "0x10");
        return 5;}    }  }
```

Classe DATA_LINK

```
int DATA_LINK(int c_type , int type, int porta) {
    char tipo[10];
    data.Local_Data_Id ++ ;
    long int IPV4_DATA_LINK;
    long double IPV6_DATA_LINK;
    long int DATA_LINK_UNNUMBERED;
    printf("entrou no data");
    lmp.obj.classe = 12;
    lmp.obj.n = 0;
    switch (c_type) {
        case 1:lmp.obj.c_type =c_type;
            IPV4_DATA_LINK = data.Local_Data_Id;
            lmp.obj.objeto[1] =IPV4_DATA_LINK;
            funcao_socket_send(porta);
            break;
        case 2:lmp.obj.c_type =c_type;
            IPV6_DATA_LINK = data.Local_Data_Id;
            lmp.obj.objeto[1] =IPV6_DATA_LINK;
            funcao_socket_send(porta);
            break;
        case 3:lmp.obj.c_type =c_type;
            DATA_LINK_UNNUMBERED = data.Local_Data_Id;
            lmp.obj.objeto[1] = DATA_LINK_UNNUMBERED;
            funcao_socket_send(porta);
            break;
        default:ERROR_CODE(2,"0x20");    }
        //if(strcmp(data.flag_dl, "0x01")==0) {/* ENLACE DE DADOS EH UMA PORTA*/}
        //else if (strcmp(data.flag_dl, "0x02")==0) {/*ENLACE DE DADOS EH PARA TRAFEGO
DE USUARIO*/}
        // else ERROR_CODE(2, "0x08");
        data.sub_obj.type = type;
        switch (data.sub_obj.type) {
            case 1: {
                lmp.obj.objeto[1] = 1;
                tipo1subobjects();
                funcao_socket_send(porta);
                break;return 1;}
```

```
case 2: {
        lmp.obj.objeto[1] = 2;
        printf("%ld", lmp.obj.objeto[2]);
        tipo2subobjects();
        funcao_socket_send(porta);
        break;return 1;}
default:ERROR_CODE(2, "0x20");
return 5;   }}
```

Tipo 1 de Sub-object: Tipo de Comutação

```
void tipo1subobjects() {
    struct tipo1 {
    int type_so1;
    int length_so1;
    char Type_switching;
    char EncType_so1;
    float Faixa_min;
    float Faixa_max;
    } tipo1;}
```

Tipo 2 de Sub-objeto: Wavelength

```
void tipo2subobjects() {
    struct tipo2 {
    int Type_so2;
    int length_so2;
    char Reserved_so2;
    long int Wavelength_so2;
    } tipo2;}
```

Classe de ERROR_CODE

```
void ERROR_CODE(int c_type, char error[4]) {
    char BEGIN_VERIFY_ERROR;
    char LINK_SUMMARY_ERROR;
    lmp.obj.classe = 20;
```

```
        if (c_type == 1) {
                strcpy(error,"BEGIN_VERIFY_ERROR");
        if (strcmp(error, "0x01")==0) {
        //verifica enlace nao suportado
                printf("Procedimento da verificação do enlace não suportado"); }
        if (strcmp(error, "0x02")==0) {
        //verificar Unwilling;
                printf ("Verificar Unwilling"); }
        if (strcmp(error, "0x04")==0) {
        //Mecanismo nao suportado pela verificacao do transporte;
                printf ("Mecanismo nao suportado pela verificacao do transporte "); }
        if (strcmp(error, "0x08")==0) {
        //Configuracao do erro link_id;
                printf ("O erro da configuração Link_Id");}
        if (strcmp(error, "0x10")==0) {
        //Desconhecido C_type do objeto;
                printf ("C-Tipo desconhecido do objeto"); }    }
    if (lmp.obj.c_type == 2) {
        strcpy(error, "LINK_SUMMARY_ERROR");
        if (strcmp(error, "0x01")==0) {
        //Parametros nao aceitaveis e nao-negociaveis do LINK_SUMMARY;
                printf ("Inaceitável parâmetros não-negociáveis de LINK_SUMMARY"); }
        if (strcmp(error, "0x02")==0) {
        //Renegociacao dos parametros do LINK_SUMMARY;
                printf ("Renegociação de parâmetros LINK_SUMMARY");  }
        if (strcmp(error, "0x04")==0) {
        //Objeto invalido do TE_LINK;
                printf ("0 objeto inválido de TE_LINK.");  }
        if (strcmp(error, "0x08")==0) {
        //Objeto invalido do DATA_LINK
                printf ("O objeto inválido de DATA_LINK."); }
        if (strcmp(error, "0x10")==0) {
        //Desconhecido C_type do objeto TE_LINK;
                printf ("C-Tipo desconhecido do objeto TE_LINK"); }
        if (strcmp(error, "0x20")==0) {
        //Desconhecido C_type do objeto DATA_LINK;
                printf ("C-Tipo desconhecido do objeto TE_LINK"); }    }}
```

APÊNDICE B - Algoritmo Baseado na MEF do Canal de Controle

Inicializar a variável ESTADO com Down;

CASO *ESTADO* SEJA:

DOWN:

SE o nó local inicia a conversa ENTÃO

Conhecer o IP do destino manualmente ou automaticamente;

Envia mensagem para negociar parâmetros;

Ir para o próximo estado (CONFSND);

SENÃO

ENQUANTO não recebe mensagem LMP do vizinho FAÇA

Decrementa o tempo de espera ate expirar;

SE expirou ENTÃO

Ir para o estado inicial (DOWN);

FIM-SE

FIM-ENQUANTO

Ir para o próximo estado (CONFSND);

FIM-SE

CONFSND:

SE recebeu uma mensagem Config do vizinho ENTÃO

SE reconhece os parâmetros ENTÃO

Ir para o próximo estado (CONFRCV);

SENÃO

Envia mensagem para negociar parâmetros;

Ir para o estado inicial (DOWN);

FIM-SE

FIM-SE

ENQUANTO espera receber mensagem de resposta do vizinho FAÇA

Decrementa o tempo de espera ate expirar;

SE expirou ENTÃO

Retransmite a mensagem Config ao vizinho;

SE houve muita retransmissões ENTÃO

Ir para o estado inicial (DOWN);

FIM-SE

FIM-SE

FIM-ENQUANTO

SE mensagem recebida reconheceu a enviada: ConfigACK ENTÃO

SE reconhece os parâmetros ENTÃO

Ir para o próximo estado (ACTIVE);

SENÃO

Envia mensagem para negociar parâmetros;

Ir para o estado inicial (DOWN);

FIM-SE

SENÃO SE mensagem recebida não reconheceu a enviada: ConfigNACK ENTÃO

Efetua um acordo de parâmetros;

SE acordo ocorreu com sucesso ENTÃO

Envia mensagem para renegociar os novos parâmetros;

Ir para o estado (CONFSND);

SENÃO

Ir para o estado inicial (DOWN);

FIM-SE

SENÃO SE o tempo recebeu uma mensagem Config ao mesmo tempo que foi emitida ao vizinho ENTÃO

SE node_id da mensagem enviada > node_id da mensagem recebida ENTÃO

Ignora Config recebida;

Ir para o estado (CONFSND);

SENÃO

Ignora Config enviada;

Ir para o estado (CONFSND);

FIM-SE

SENÃO

Ir para o estado inicial (DOWN);

FIM SE;

CONFRCV:

SE mensagem recebida reconheceu a enviada: ConfigACK ENTÃO

SE reconhece os parâmetros ENTÃO

Ir para o próximo estado (ACTIVE);

SENÃO

Envia mensagem para negociar parâmetros;

Ir para o estado inicial (DOWN);

FIM-SE

SENÃO SE mensagem recebida não reconheceu a enviada: ConfigNACK ENTÃO

Efetua um acordo de parâmetros;

SE acordo ocorreu com sucesso ENTÃO

Envia mensagem para renegociar os novos parâmetros;

Ir para o estado (CONFSND);

SENÃO

Ir para o estado inicial (DOWN);

FIM-SE

SENÃO SE administrador pede que o canal de controle vá para down ENTÃO

Ir para o estado de ajuste (GOIG DOWN);

SENÃO SE verificar que o canal não está mais disponível ENTÃO

Ir para o estado inicial (DOWN);

SENÃO

Ir para o estado inicial (DOWN);

FIM SE;

ACTIVE:

SE administrador pede que o canal de controle vá para down ENTÃO

Ir para o estado de ajuste (GOIG DOWN);

SENÃO SE verificar que o canal não está mais disponível ENTÃO

Ir para o estado inicial (DOWN);

SENÃO SE recebeu uma mensagem com o flag ControlChannelDown ativo ENTÃO

Ir para o estado inicial (DOWN);

FIM-SE

// Neste estado há envio e recebimentos de mensagem Hello que serve para manter a conectividade do canal e detectar falhas rapidamente.

// O tempo do HelloDeadInterval expira indicando que nenhum pacote Hello foi recebido. Isto move o canal de controle novamente na negociação do estado, e dependendo da configuração local, isto provocará: a) a emissão de mensagens Config periódicas ou b) um período da espera para receber mensagens Config do nó remoto

```
HelloInterval <- 150ms;
HelloDeadInterval =<-5500ms;
TxSeqNum <- 1;
RcvSeqNum <- 0;

ENQUANTO não recebe mensagens Hello FAÇA
    Decrementa o tempo de HelloInterval;
        SE tempo de HelloInterval expirar ENTÃO
            Envia mensagem Hello;
        SENÃO
            Decrementa o tempo de HelloDeadInterval
            SE HelloDeadInterval expirar ENTÃO
                ENQUANTO espera receber mensagem de resposta do vizinho FAÇA
                    Decrementa o tempo de espera;
                    SE expirou ENTÃO
                        Retransmite a mensagem Config ao vizinho;
                        SE houve muita retransmissões ENTÃO
                          Ir para o estado inicial (DOWN);
                        FIM-SE
                    FIM-SE
                FIM-ENQUANTO
            FIM-SE
        FIM-SE
FIM-ENQUANTO

SE recebeu mensagem HELLO ENTÃO
    Verifica SeqNum;
        SE SeqNum foi o esperado ENTÃO
            Ir para o próximo estado (UP);
        SENÃO
            Rejeita HELLO;
        FIM-SE

SE SENÃO mensagem recebida reconheceu a enviada: ConfigACK ENTÃO
    SE reconhece os parâmetros ENTÃO
        Ir para o próximo estado (ACTIVE);
    SENÃO
        Envia mensagem para negociar parâmetros;
        Ir para o estado inicial (DOWN);
    FIM-SE
```

SENÃO SE mensagem recebida não reconheceu a enviada: ConfigNACK ENTÃO

Efetua um acordo de parâmetros;

SE acordo ocorreu com sucesso ENTÃO

Envia mensagem para renegociar os novos parâmetros;

Ir para o estado (CONFSND);

SENÃO

Ir para o estado inicial (DOWN);

FIM-SE

SENÃO

Ir para o estado inicial (DOWN);

FIM SE;

UP:

SE administrador pede que o canal de controle vá para down ENTÃO

Ir para o estado de ajuste (GOIG DOWN);

SENÃO SE verificar que o canal não está mais disponível ENTÃO

Ir para o estado inicial (DOWN);

SENÃO SE recebeu uma mensagem com o flag ControlChannelDown ativo ENTÃO

Ir para o estado inicial (DOWN);

FIM-SE

// Neste estado há envio e recebimentos de mensagem Hello que serve para manter a conectividade do canal e detectar falhas rapidamente.

// O tempo do HelloDeadInterval expira indicando que nenhum pacote Hello foi recebido. Isto move o canal de controle novamente na negociação do estado, e dependendo da configuração local, isto provocará: a) a emissão de mensagens Config periódicas ou b) um período da espera para receber mensagens Config do nó remoto

HelloInterval <- 150ms;

HelloDeadInterval =<-5500ms;

TxSeqNum <- 1;

RcvSeqNum <- 0;

ENQUANTO não recebe mensagens Hello FAÇA

Decrementa o tempo de HelloInterval;

SE tempo de HelloInterval expirar ENTÃO

Envia mensagem Hello;

SENÃO

Decrementa o tempo de HelloDeadInterval

SE HelloDeadInterval expirar ENTÃO

ENQUANTO espera receber mensagem de resposta do vizinho FAÇA

Decrementa o tempo de espera;

SE expirou ENTÃO

Retransmite a mensagem Config ao vizinho;

SE houve muita retransmissões ENTÃO

Ir para o estado inicial (DOWN);

FIM-SE

FIM-SE

FIM-ENQUANTO

FIM-SE

FIM-SE

FIM-ENQUANTO

SE recebeu mensagem HELLO ENTÃO

Verifica SeqNum;

SE SeqNum foi o esperado ENTÃO

Ir para o próximo estado (UP);

SENÃO

Rejeita HELLO;

FIM-SE

SE SENÃO mensagem recebida reconheceu a enviada: ConfigACK ENTÃO

SE reconhece os parâmetros ENTÃO

Ir para o próximo estado (ACTIVE);

SENÃO

Envia mensagem para negociar parâmetros;

Ir para o estado inicial (DOWN);

FIM-SE

SENÃO SE mensagem recebida não reconheceu a enviada: ConfigNACK ENTÃO

Efetua um acordo de parâmetros;

SE acordo ocorreu com sucesso ENTÃO

Envia mensagem para renegociar os novos parâmetros;

Ir para o estado (CONFSND);

SENÃO

Ir para o estado inicial (DOWN);

FIM-SE

SENÃO

Ir para o estado inicial (DOWN);

FIM SE;

GOING DOWN:

ENQUANTO não recebe mensagens com o flag ControlChannelDown ajustado FAÇA

Decrementa contador de tempo;

SE tempo expirou ENTÃO

Ir para o estado inicial (DOWN);

FIM-SE

FIM-ENQUANTO

SE recebeu uma mensagem com o flag ControlChannelDown ativo ENTÃO

Ir para o estado inicial (DOWN);

SENÃO

Ir para o estado inicial (DOWN);

FIM SE;

FIM CASO;

ESTADO < - PROXIMO_ESTADO;

FIM.

APÊNDICE C - Algoritmo Baseado na MEF do Enlace TE

Inicializar a variável ESTADO2 com Down;

CASO *ESTADO2* SEJA:

DOWN:

Enlace TE <- um ou mais canais de dados habilitados;
Ir para o próximo estado (INIT);

INIT:

SE o nó local inicia a conversa ENTÃO
Envia mensagem para negociar parâmetros;
Ir para o próximo estado (UP);

SENÃO SE recebeu uma mensagem LinkSummary do vizinho ENTÃO
SE reconhece os parâmetros ENTÃO
Ir para o próximo estado (UP);
SENÃO
Envia mensagem para negociar parâmetros;
Ir para o estado inicial (INIT);
FIM-SE
FIM-SE

ENQUANTO espera receber mensagem de resposta do vizinho FAÇA
Decrementa o tempo de espera ate expirar;
SE expirou ENTÃO
Retransmite a mensagem LinkSummary ao vizinho;
Ir para o próximo estado (INIT);
SE houve muita retransmissões ENTÃO
Ir para o estado inicial (DOWN);
FIM-SE
FIM-SE
FIM-ENQUANTO

SE mensagem recebida reconheceu a enviada: LinkSummaryACK ENTÃO
SE reconhece os parâmetros ENTÃO
Ir para o próximo estado (UP);

SENÃO

Envia mensagem para negociar parâmetros;

Ir para o estado inicial (INIT);

FIM-SE

SENÃO SE mensagem recebida não reconheceu a enviada: LinkSummaryNACK ENTÃO

Efetua um acordo de parâmetros;

SE acordo ocorreu com sucesso ENTÃO

Envia mensagem para renegociar os novos parâmetros;

Ir para o estado (INIT);

SENÃO

Ir para o estado inicial (DOWN);

FIM-SE

SENÃO SE o primeiro canal de controle ativo vai para up ENTÃO

Ir para o próximo estado (INIT);

SENÃO SE o último canal de controle ativo o controle vai para down ENTÃO

Ir para o próximo estado (INIT);

SENÃO SE o último canal de dados do enlace TE foi removida ENTÃO

Ir para o próximo estado (DOWN);

SENÃO

Ir para o próximo estado (DOWN);

FIM-SE;

UP:

SENÃO SE recebeu uma mensagem LinkSummary do vizinho ENTÃO

SE reconhece os parâmetros ENTÃO

Ir para o próximo estado (UP);

SENÃO

Envia mensagem para negociar parâmetros;

Ir para o estado inicial (UP);

FIM-SE

ENQUANTO espera receber mensagem de resposta do vizinho FAÇA

Decrementa o tempo de espera ate expirar;

SE expirou ENTÃO

Retransmite a mensagem LinkSummary ao vizinho;

Ir para o próximo estado (UP);

SE houve muita retransmissões ENTÃO

Ir para o estado inicial (DOWN);

FIM-SE

FIM-SE

FIM-ENQUANTO

SE mensagem recebida reconheceu a enviada: LinkSummaryACK ENTÃO

SE reconhece os parâmetros ENTÃO

Ir para o próximo estado (UP);

SENÃO

Envia mensagem para negociar parâmetros;

Ir para o estado inicial (UP);

FIM-SE

SENÃO SE mensagem recebida não reconheceu a enviada: LinkSummaryNACK ENTÃO

Efetua um acordo de parâmetros;

SE acordo ocorreu com sucesso ENTÃO

Envia mensagem para renegociar os novos parâmetros;

Ir para o estado (UP);

SENÃO

Ir para o estado inicial (UP);

FIM-SE

SENÃO SE o último canal de controle ativo o controle vai para down ENTÃO

Ir para o próximo estado (DEG);

SENÃO SE o último canal de dados do enlace TE foi removida ENTÃO

Ir para o próximo estado (UP);

SENÃO

Ir para o próximo estado (DOWN);

FIM-SE;

DEG:

SENÃO SE o primeiro canal de controle ativo vai para up ENTÃO

Ir para o próximo estado (UP);

SENÃO SE o último canal de dados do enlace TE foi removida ENTÃO

Ir para o próximo estado (DOWN);

SENÃO

Ir para o próximo estado (DOWN);

FIM-SE;

FIM CASO;

ESTADO2<-PROXIMO_ESTADO;

FIM.

APÊNDICE D – Modificações DRAGON.SH

```
#!/bin/sh

if test "$DRAGON_PREFIX" = ""; then
    PREFIX=/usr/local/dragon
else
    PREFIX=$DRAGON_PREFIX
fi

ETC_DIR=$PREFIX/etc

ZEBRA_DAEMON=$PREFIX/sbin/zebra
ZEBRA_ARGS="-d -f $ETC_DIR/zebra.conf"

OSPF_DAEMON=$PREFIX/sbin/ospfd
OSPF_ARGS="-d -f $ETC_DIR/ospfd.conf"

RSVP_DAEMON=$PREFIX/bin/RSVPD
RSVP_ARGS="-c $ETC_DIR/RSVPD.conf -d -o /var/log/RSVPD.log -L
select,ref,packet,timer"

DRAGON_DAEMON=$PREFIX/bin/dragon
DRAGON_ARGS="-d -f $ETC_DIR/dragon.conf"

NODE_AGENT=$PREFIX/bin/node_agent
NODE_AGENT_ARGS="-d -c $ETC_DIR/node_agent.conf"

# for NARBs we need 2 ospfd instances.  the above
# OSPF_DAEMON/OSPF_ARGS variables are not used for NARBs:

OSPF_INTER_DAEMON=$PREFIX/sbin/ospfd
OSPF_INTER_ARGS="-d -I -P 2614 -f $ETC_DIR/ospfd-inter.conf"

OSPF_INTRA_DAEMON=$PREFIX/sbin/ospfd
OSPF_INTRA_ARGS="-d -P 2604 -f $ETC_DIR/ospfd-intra.conf"

NARB_DAEMON=$PREFIX/bin/run_narb.sh
NARB_ARGS=""

LMPCC_DAEMON=$PREFIX/sbin/lmpcc
LMPCC_ARGS=""

LMPETE_DAEMON=$PREFIX/sbin/lmpete
LMPETE_ARGS=""

#
# need to have a way for 'stop' to recognize/kill both ospfd
instances...
#

#
# See which daemons are already running...
#

case "`uname`" in
        Linux* | *BSD* | Darwin*)
                zebra_pid=`ps  axwww | grep -v awk | awk '{if
(match($5, ".*/zebra$")       || $5 == "zebra")  print $1}'`
```

```
                rsvp_pid=`ps   axwww | grep -v awk | awk '{if
(match($5, ".*/RSVPD$")       || $5 == "RSVPD")  print $1}'`
                dragon_pid=`ps axwww | grep -v awk | awk '{if
(match($5, ".*/dragon$")      || $5 == "dragon") print $1}'`
                node_agent_pid=`ps axwww | grep -v awk | awk '{if
(match($5, ".*/node_agent$")      || $5 == "node_agent") print $1}'`
                narb_pid=`ps   axwww | grep -v awk | awk '{if
(match($5, ".*/narb$")        || $5 == "narb")   print $1}'`
                rce_pid=`ps   axwww | grep -v awk | awk '{if (match($5,
".*/rce$")        || $5 == "rce")   print $1}'`
                telnet_pid=`ps   axwww | grep -v awk | awk '{if ($5 ==
match($5, ".*/telnet$") ||   "telnet")   print $1}'`
        #lmpcc_pid=`ps   axwww | grep -v awk | awk '{if ($5 ==
match($5, ".*/lmpcc$") ||   "lmpcc")   print $1}'`
        lmpcc_pid=`ps axwww | grep -v awk | awk '{if (match($0,
".*/lmpcc")) print $1}'`
        #lmpete_pid=`ps   axwww | grep -v awk | awk '{if ($5 ==
match($5, ".*/lmpete$") ||   "lmpete")   print $1}'`
        lmpete_pid=`ps axwww | grep -v awk | awk '{if (match($0,
".*/lmpete")) print $1}'`
                # XXX ugh...there must be a better way to do
this...this is a kludge
                # maybe search for OSPF_INTER_ARGS and
OSPF_INTRA_ARGS...or OSPF_ARGS?
                ospf_intra_pid=`ps axwww | grep -v awk | awk '{if
(match($0, ".*/ospfd-intra")  || match($0, ".*/ospfd.*conf")) print
$1}'`
                ospf_inter_pid=`ps axwww | grep -v awk | awk '{if
(match($0, ".*/ospfd-inter")) print $1}'`
                ;;
        *)
                zebra_pid=""
                ospf_intra_pid=""
                ospf_inter_pid=""
                rsvp_pid=""
                dragon_pid=""
                node_agent_pid=""
                narb_pid=""
                rce_pid=""
                telnet_pid=""
        lmpcc_pid=""
        lmpete_pid=""
                ;;
esac

#
# Start or stop the daemons based upon the first argument to the
script.
#

case $1 in
    start-vlsr | startvlsr | restart-vlsr)
        echo "dragon-sw: turning on coredump with unlimited core size."
        ulimit -c unlimited

        # XXX for CLI based switch control only
        if test "$telnet_pid" != ""; then
        killall -9 telnet
```

```
fi

    if test "$zebra_pid" != ""; then
    kill $zebra_pid
fi
$ZEBRA_DAEMON $ZEBRA_ARGS
if test $? != 0; then
    echo "dragon-sw: unable to start zebra daemon."
    exit 1
fi
echo "dragon-sw: started zebra daemon."

if test "$lmpcc_pid" != ""; then
    kill $lmpcc_pid
fi
$LMPCC_DAEMON $LMPCC_ARGS
if test $? != 0; then
    echo "dragon-sw: unable to start lmpcc daemon."
    exit 1
fi
echo "dragon-sw: started lmpcc daemon."

if test "$lmpete_pid" != ""; then
    kill $lmpete_pid
fi
$LMPETE_DAEMON $LMPETE_ARGS
if test $? != 0; then
    echo "dragon-sw: unable to start lmpete daemon."
    exit 1
fi
echo "dragon-sw: started lmpete daemon."

    # XXX again, a bit of a hack here...
    if test "$ospf_intra_pid" != ""; then
    kill $ospf_intra_pid
fi
$OSPF_DAEMON $OSPF_ARGS
if test $? != 0; then
    echo "dragon-sw: unable to start ospf daemon."
    exit 1
fi
echo "dragon-sw: started ospf daemon."

    if test "$rsvp_pid" != ""; then
    kill $rsvp_pid
fi
$RSVP_DAEMON $RSVP_ARGS
if test $? != 0; then
    echo "dragon-sw: unable to start rsvp daemon."
    exit 1
fi
echo "dragon-sw: started rsvp daemon."

    echo "sleeping for 2 seconds before starting dragon daemon..."
    sleep 2

    if test "$dragon_pid" != ""; then
    kill $dragon_pid
```

```
fi
$DRAGON_DAEMON $DRAGON_ARGS
if test $? != 0; then
    echo "dragon-sw: unable to start dragon daemon."
    exit 1
fi
echo "dragon-sw: started dragon daemon."
;;

# XXX running software in uni mode w/o zebra & ospfd
start-uni | startuni | restart-uni)
    echo "dragon-sw: turning on coredump with unlimited core size."
    ulimit -c unlimited

    echo "dragon-sw: starting under UNI mode."
    echo ""
    if test "$rsvp_pid" != ""; then
        kill $rsvp_pid
    fi
    $RSVP_DAEMON $RSVP_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start rsvp daemon."
        exit 1
    fi
    echo "dragon-sw: started rsvp daemon."

    echo "sleeping for 2 seconds before starting dragon daemon..."
    sleep 2

    if test "$dragon_pid" != ""; then
        kill $dragon_pid
    fi
    $DRAGON_DAEMON $DRAGON_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start dragon daemon."
        exit 1
    fi
    echo "dragon-sw: started dragon daemon."

    if test "$node_agent_pid" != ""; then
        kill $node_agent_pid
    fi
$NODE_AGENT $NODE_AGENT_ARGS
if test $? != 0; then
    echo "dragon-sw: unable to start node agent."
    exit 1
fi
echo "dragon-sw: started node agent."
;;

start-narb | startnarb | restart-narb)
    echo "dragon-sw: turning on coredump with unlimited core size."
    ulimit -c unlimited

    if test "$narb_pid" != ""; then
    kill $narb_pid
    echo "dragon-sw: stopped narb daemon."
```

```
            echo "Waiting 5 seconds for NARB to cleanup domain topology
..."
            sleep 5
    fi

        if test "$rce_pid" != ""; then
        kill $rce_pid
        echo "dragon-sw: stopped rce daemon."
    fi

        if test "$zebra_pid" != ""; then
        kill $zebra_pid
    fi
    $ZEBRA_DAEMON $ZEBRA_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start zebra daemon."
        exit 1
    fi
    echo "dragon-sw: started zebra daemon."

    if test "$lmpcc_pid" != ""; then
        kill $lmpcc_pid
    fi
    $LMPCC_DAEMON $LMPCC_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start lmpcc daemon."
        exit 1
    fi
    echo "dragon-sw: started lmpcc daemon."

    if test "$lmpete_pid" != ""; then
        kill $lmpete_pid
    fi
    $LMPETE_DAEMON $LMPETE_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start lmpete daemon."
        exit 1
    fi
    echo "dragon-sw: started lmpete daemon."

        if test "$ospf_inter_pid" != ""; then
        kill $ospf_inter_pid
    fi
    $OSPF_INTER_DAEMON $OSPF_INTER_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start ospf inter-domain daemon."
        exit 1
    fi
    echo "dragon-sw: started ospf inter-domain daemon."

        if test "$ospf_intra_pid" != ""; then
        kill $ospf_intra_pid
    fi
    $OSPF_INTRA_DAEMON $OSPF_INTRA_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start ospf intra-domain daemon."
        exit 1
    fi
```

```
    echo "dragon-sw: started ospf intra-domain daemon."

        echo "sleeping for 5 seconds before starting narb (rce & narb
daemons)...please stand by."
        sleep 5

    $NARB_DAEMON $NARB_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start narb daemon."
        exit 1
    fi
    echo "dragon-sw: started narb daemons."
    ;;

    start-vlsr-narb | restart-vlsr-narb)
        echo "dragon-sw: turning on coredump with unlimited core size."
        ulimit -c unlimited

        if test "$narb_pid" != ""; then
        kill $narb_pid
        echo "dragon-sw: stopped narb daemon."
            echo "Waiting 5 seconds for NARB to cleanup domain topology
..."
            sleep 5
    fi
        if test "$rce_pid" != ""; then
        kill $rce_pid
        echo "dragon-sw: stopped rce daemon."
    fi

        # XXX for CLI based switch control only
        if test "$telnet_pid" != ""; then
            killall -9 telnet
        fi

        if test "$zebra_pid" != ""; then
        kill $zebra_pid
    fi
    $ZEBRA_DAEMON $ZEBRA_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start zebra daemon."
        exit 1
    fi
    echo "dragon-sw: started zebra daemon."

        if test "$lmpcc_pid" != ""; then
        kill $lmpcc_pid
    fi
    $LMPCC_DAEMON $LMPCC_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start lmpcc daemon."
        exit 1
    fi
    echo "dragon-sw: started lmpcc daemon."
#
        if test "$lmpete_pid" != ""; then
        kill $lmpete_pid
    fi
```

```
    $LMPETE_DAEMON $LMPETE_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start lmpete daemon."
        exit 1
    fi
    echo "dragon-sw: started lmpete daemon."

        if test "$ospf_inter_pid" != ""; then
        kill $ospf_inter_pid
    fi
    $OSPF_INTER_DAEMON $OSPF_INTER_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start ospf inter-domain daemon."
        exit 1
    fi
    echo "dragon-sw: started ospf inter-domain daemon."

        if test "$ospf_intra_pid" != ""; then
        kill $ospf_intra_pid
    fi
    if test -e $ETC_DIR/ospfd.conf; then
            $OSPF_DAEMON $OSPF_ARGS
    elif test -e $ETC_DIR/ospfd-intra.conf; then
        $OSPF_INTRA_DAEMON $OSPF_INTRA_ARGS
    else
        false
    fi
    if test $? != 0; then
        echo "dragon-sw: unable to start ospf intra-domain daemon."
        exit 1
    fi
    echo "dragon-sw: started ospf intra-domain daemon."

        if test "$rsvp_pid" != ""; then
            kill $rsvp_pid
        fi
        $RSVP_DAEMON $RSVP_ARGS
        if test $? != 0; then
            echo "dragon-sw: unable to start rsvp daemon."
            exit 1
        fi
        echo "dragon-sw: started rsvp daemon."

        echo "sleeping for 2 seconds before starting dragon daemon..."
        sleep 2

        if test "$dragon_pid" != ""; then
            kill $dragon_pid
        fi
        $DRAGON_DAEMON $DRAGON_ARGS
        if test $? != 0; then
            echo "dragon-sw: unable to start dragon daemon."
            exit 1
        fi
        echo "dragon-sw: started dragon daemon."

        echo "sleeping for 5 seconds before starting narb (rce & narb
daemons)...please stand by."
```

```
        sleep 5

    $NARB_DAEMON $NARB_ARGS
    if test $? != 0; then
        echo "dragon-sw: unable to start narb daemon."
        exit 1
    fi
    echo "dragon-sw: started narb daemons."
    ;;

    stop)
        if test "$telnet_pid" != ""; then
        killall -9 telnet
    fi

        if test "$narb_pid" != ""; then
        kill $narb_pid
        echo "dragon-sw: stopped narb daemon."
            echo "Waiting 5 seconds for NARB to cleanup domain topology
..."
            sleep 5
    fi

        if test "$rce_pid" != ""; then
        kill $rce_pid
        echo "dragon-sw: stopped rce daemon."
    fi

        if test "$zebra_pid" != ""; then
        kill $zebra_pid
        echo "dragon-sw: stopped zebra daemon."
    fi
    #
        if test "$lmpcc_pid" != ""; then
        kill $lmpcc_pid
        echo "dragon-sw: stopped lmpcc daemon."
    fi
    #
        if test "$lmpete_pid" != ""; then
        kill $lmpete_pid
        echo "dragon-sw: stopped lmpete daemon."
    fi
        if test "$rsvp_pid" != ""; then
        kill $rsvp_pid
        echo "dragon-sw: stopped rsvp daemon."
    fi

        if test "$ospf_intra_pid" != ""; then
        kill $ospf_intra_pid
        echo "dragon-sw: stopped intra-domain ospf daemon."
    fi

        if test "$ospf_inter_pid" != ""; then
        kill $ospf_inter_pid
        echo "dragon-sw: stopped inter-domain ospf daemon."
    fi

        if test "$dragon_pid" != ""; then
```

```
        kill $dragon_pid
        echo "dragon-sw: stopped dragon daemon."
    fi

        if test "$node_agent_pid" != ""; then
        kill $node_agent_pid
        echo "dragon-sw: stopped node agent."
    fi
    ;;

    status)
        if test "$zebra_pid" != ""; then
        echo "dragon-sw: zebra daemon is running, pid=$zebra_pid."
    else
        echo "dragon-sw: zebra daemon is NOT running."
    fi
    #
        if test "$lmpcc_pid" != ""; then
        echo "dragon-sw: lmpcc daemon is running, pid=$lmpcc_pid."
    else
        echo "dragon-sw: lmpcc daemon is NOT running."
    fi
    #
        if test "$lmpete_pid" != ""; then
        echo "dragon-sw: lmpete daemon is running, pid=$lmpete_pid."
    else
        echo "dragon-sw: lmpete daemon is NOT running."
    fi
        if test "$rsvp_pid" != ""; then
        echo "dragon-sw: rsvp daemon is running, pid=$rsvp_pid."
    else
        echo "dragon-sw: rsvp daemon is NOT running."
    fi

        if test "$ospf_inter_pid" != ""; then
        echo "dragon-sw: inter-domain ospf daemon is running,
pid=$ospf_inter_pid."
    fi

        if test "$ospf_intra_pid" != ""; then
        echo "dragon-sw: intra-domain ospf daemon is running,
pid=$ospf_intra_pid."
    fi

        if test "$dragon_pid" != ""; then
        echo "dragon-sw: dragon daemon is running, pid=$dragon_pid."
    else
        echo "dragon-sw: dragon daemon is NOT running."
    fi

        if test "$node_agent_pid" != ""; then
        echo "dragon-sw: node agent is running, pid=$node_agent_pid."
    else
        echo "dragon-sw: node agent is NOT running."
    fi

        if test "$narb_pid" != ""; then
        echo "dragon-sw: narb daemon is running, pid=$narb_pid."
```

```
    else
        echo "dragon-sw: narb daemon is NOT running."
    fi

        if test "$rce_pid" != ""; then
        echo "dragon-sw: rce daemon is running, pid=$rce_pid."
    else
        echo "dragon-sw: rce daemon is NOT running."
    fi
    ;;

    *)
        echo "Usage: $0 {start-vlsr|restart-vlsr|start-uni|restart-
uni|start-narb|restart-narb|start-vlsr-narb|restart-vlsr-
narb|status|stop}"
    exit 1
    ;;
esac

#
# Exit with no errors.
#

exit 0
```