

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO**

**UMA SOLUÇÃO DE AUTENTICAÇÃO
FIM A FIM PARA O LDP
(*LABEL DISTRIBUTION PROTOCOL*)**

AUTOR: Leonardo Neves Bernardo

ORIENTADOR: Carlos Becker Westphall

BANCA EXAMINADORA: Carlos Becker Westphall

Morvan Daniel Müller

Carla Merkle Westphall

PALAVRAS-CHAVE: LDP, MPLS, *autenticação, integridade, segurança, TCP/IP, roteamento.*

Florianópolis, Junho de 2003.

DEDICAÇÕES

Dedico este trabalho a todos que de alguma forma ajudam a construir, estender e aperfeiçoar softwares e protocolos de domínio público colaborando assim para a evolução científica e tecnológica da computação.

AGRADECIMENTOS

Agradeço, em especial, à Morvan Daniel Müller pelas colaborações na fase de implementação, explicações técnicas, sua paciência e motivação. Sua colaboração foi essencial para o andamento do trabalho.

Gostaria também de prestar o meu reconhecimento pela grande colaboração prestada pelo colega Bruno Bisol que colaborou de forma significativa na utilização das funções de criptografia da biblioteca openssl.

Agradeço ao Prof. Dr. Carlos Becker Westphall, à Profa. Dra. Carla Merkle Westphall e ao Prof. Dr. Ricardo Felipe Custódio por revisões, sugestões e contribuições ao projeto e à monografia.

Finalmente, agradeço ao Sr. James Leu, desenvolvedor do projeto Mpls-Linux da sourceforge.net e ao Sr. Jeremy De Clercq, do grupo de Estratégias de Redes da empresa Alcatel da Bélgica. Suas contribuições à Morvan Daniel Müller foram fundamentais para o andamento do projeto.

SÍMBOLOS E ABREVIações

A seguir são apresentados os principais símbolos e abreviações usados neste trabalho:

BGP	<i>Border Gateway Protocol</i>
CoS	<i>Class of Services</i>
CR-LDP	<i>Constraint-based Routed Label Distribution Protocol</i>
FEC	<i>Forward Equivalence Class</i>
IETF	<i>Internet Engineering Task Force</i>
IPSec	<i>IP Security Standard</i>
IPV4	IP Versão 4
ISO	International Standards Organization
LDP	<i>Label Distribution Protocol</i>
LER	<i>Label Edge Router</i>
LIB	<i>Label Information Base</i>
LSP	Label Switch Path
LSR	Label Switch Router
MAC	Message Authentication Code
MPLS	<i>Multi-Protocol Label Switching</i>
NTP	<i>Network Time Protocol</i>
NIST	<i>National Institute for Standards and Technology</i>
PDU	<i>Protocol Data Unit</i>
PKI	<i>Public Key Infrastructure</i>
RIP	<i>Routing Information Protocol</i>
OSI	Open Systems Interconnection
OSPF	<i>Open Shortest Path First</i>
QoS	<i>Quality of Services</i>
RSVP	<i>Resource Reservation Protocol</i>
SSL	<i>Socket Security Layer</i>
TCP	<i>Transmission Control Protocol</i>
TE	<i>Traffic Engineering</i>
TLS	<i>Transport Layer Security</i>
TLV	<i>Type-Length-Value</i>
TTL	<i>Time to Live</i>
VPN	<i>Virtual Private Networks</i>

SUMÁRIO

1 - Introdução.....	1
1.1 - Tema	1
1.2 - Objetivo Geral.....	1
1.3 - Objetivo Específico	1
1.4 - Justificativas	2
1.5 - Problema	2
2 - Desenvolvimento do Trabalho	4
2.1 - Revisão Bibliográfica	4
2.1.1 - MPLS	4
2.1.1.1 - Plano de encaminhamento	7
2.1.1.2 - Plano de controle.....	9
2.1.2 – LDP	12
2.1.2.1 - Funcionamento do LDP	13
2.1.2.2 - PDUs LDP e TLVs.....	16
2.1.2.3 - Gerência de Rótulos	18
2.1.3 – TCP/IP	21
2.1.3.1 - Protocolo IP	22
2.1.3.2 - TCP	25
2.1.4 – LINUX.....	25
2.1.5 – MPLS-LINUX.....	26
2.1.6 – ZEBRA	26
2.1.7 – LDP-PORTABLE	27
2.1.8 – Criptografia Assimétrica	27
2.1.8.1 - RSA	28
2.1.9 – Funções Hash	29
2.2 – Problemas de Segurança do LDP.....	30
2.3 – Autenticação Fim a Fim para o LDP	31
2.3.1 - Proposta de Autenticação Fim a Fim para o LDP	35
2.3.2 - Modelo de Autenticação Proposto	37
2.3.3 - Novos TLVs Definidos.....	39
2.3.3.1 - TLV de Hash.....	40
2.3.3.2 - TLV de Nonce.....	41
2.3.3.3 - Novo Código de Status "Authentication Failed"	41
2.3.3.4 - Considerações sobre o Registro dos TLVs	42

2.3.4 - Procedimentos da Autenticação Fim a Fim.....	42
2.3.4.1 - Procedimentos de Envio de Mensagens LDP	43
2.3.4.2 - Procedimentos ao RECEBER Mensagens LDP	44
2.3.5 - Discussão sobre a Solução de Autenticação Proposta.....	45
2.3.5.1 - Distribuição de Chaves.....	46
2.3.5.2 - Integridade da Mensagem.....	47
2.3.5.3 - Autenticação da Origem	47
2.3.5.4 - Controle contra Ataques de Repetição.....	47
2.3.6 - Considerações sobre Algoritmos de Criptografia	48
2.3.7 - Distribuição de Chaves usando Certificação Digital.....	48
2.4 - Contribuições Efetivas	49
2.4.2 - Implementação da Solução de Autenticação	50
2.4.2 - Validação da proposta	69
2.4.3 - Configuração e Execução no Ambiente de Testes	70
2.4.3.1 - Interface de Gerenciamento	71
2.4.4 - Resultados Obtidos.....	74
4 – Conclusões	89
5 – Referências Bibliográficas	90

RESUMO

Este trabalho tem como finalidade a demonstração da implementação de uma solução de autenticação fim a fim para o protocolo LDP (Label Distribution Protocol).

Na solução de autenticação existente no protocolo LDP não há a possibilidade de autenticação em um escopo fim-a-fim, ou seja cada LSR confia no LSR vizinho para o estabelecimento de um LSP. Nesta solução, um LSR A pode criar um LSP com seu LSR adjacente B de forma segura. O LSR B, sendo adjacente ao LSR C também pode criar um LSP de forma segura. Neste caso criou-se um LSP entre os LSR A e C, sendo que A não confia em C, portanto a solução não supre a necessidade de algumas aplicações.

Morvan Daniel Müller propôs uma solução onde um LSR de ingresso pode ser configurado de forma que um LSP somente seja criado se confiar no LSR de egresso. A autenticação aplica-se principalmente a LSRs não adjacentes, porém também pode ser usado para LSRs adjacentes.

A segurança da solução garante autenticação e integridade. A autenticação é garantida com um mecanismo de assinatura digital e a integridade dos dados é garantida com criptografia unidirecional (hash). Os ataques de repetição são protegidos com a inserção de um campo de nonce às mensagens do TLV.

Para provar a viabilidade da solução foi implementado um protótipo de rede MPLS com LDP modificado para a nova forma de autenticação. Para criar o protótipo, foi necessário o conhecimento dos protocolos envolvidos (MPLS, LDP, TCP/IP) e das ferramentas de controle da rede simulada (Linux, Zebra). O trabalho maior, no entanto, foi o empreendido na alteração do código fonte de uma implementação de código aberto do protocolo LDP (ldp-portable).

1 - INTRODUÇÃO

1.1 - TEMA

O LDP (RFC 3036) é o principal protocolo de roteamento do protocolo MPLS (RFC 3031). Ele é responsável pelo estabelecimento e liberação dos LSP, ou caminhos virtuais de um domínio MPLS. Falhas na segurança do LDP podem comprometer todo o ambiente, principalmente porque LSPs podem ser criados através de LSRs não confiáveis, possibilitando interrupção, interceptação, modificação e fabricação da informação que trafega no domínio MPLS.

Foi definida uma forma de autenticação para o LDP [ANDERSSON, 2001], porém esta solução está restrita a LSRs adjacentes e depende de uma conexão TCP entre os LSRs envolvidos. Sendo assim, esta solução não trata situações em que dois LSRs, não-adjacentes, pretendem se autenticar mutuamente fim a fim, durante o estabelecimento de um novo LSP [Müller, 2002].

A alternativa de autenticação, proposta por Morvan Daniel Müller, define uma forma de autenticação fim a fim para o protocolo LDP, além disso não depende de uma conexão TCP entre os LSR para a criação do LSP, o que comprometeria a segurança.

A solução prevê autenticação, integridade e proteção contra ataques de repetição. Pode-se assim garantir uma melhor segurança para o domínio MPLS.

Em diversas situações, onde múltiplos domínios MPLS se interconectam e necessitam se autenticar para a criação de rotas, esta nova solução de autenticação é a única que garante a segurança no tráfego dos dados através dos domínios.

1.2 - OBJETIVO GERAL

O objetivo geral deste trabalho é implementar melhorias de segurança no protocolo LDP, e conseqüentemente no protocolo MPLS.

1.3 - OBJETIVO ESPECÍFICO

O objetivo específico é de participar da implementação da solução de autenticação proposta na dissertação de Morvan Daniel Müller. A dissertação foi submetida a banca avaliadora do Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Santa Catarina.

1.4 - JUSTIFICATIVAS

O tecnologia MPLS foi projetada para solucionar diversos problemas existentes em redes IP tradicionais. Dentre outras características, destaca-se por ser mais rápido e possibilitar o tráfego com qualidade de serviço (QoS). O MPLS é ideal para grandes backbones e se adapta as redes físicas já existentes.

Apesar do MPLS já ser bastante utilizado, grande parte das definições de rotas é feito através de configuração manual em cada roteador. Com o aumento das redes MPLS e conseqüentemente de uma maior dificuldade de manutenção de rotas, cresce também a necessidade de utilização de protocolos para a distribuição de rotas de forma dinâmica. O protocolo LDP é um dos protocolos de roteamento dinâmico que suporta o protocolo MPLS.

A especificação do protocolo LDP prevê autenticação entre dois roteadores adjacentes, porém não prevê autenticação entre roteadores não adjacentes. Este trabalho faz parte da implementação de uma solução de autenticação para o LDP que suporte autenticação entre roteadores não adjacentes.

1.5 - PROBLEMA

Para provar a viabilidade da solução de autenticação proposta fez-se necessário implementar ou simular o protocolo com suas modificações na autenticação.

Optou-se por provar a solução implementando as modificações no software ldp-portable. O ldp-portable é um módulo do projeto de domínio público mpls-linux que é hospedado na sourceforge.net e implementa os protocolos MPLS e LDP para plataforma linux. O ldp-portable adiciona funcionalidade MPLS ao software de roteamento zebra, disponível em zebra.org, também de domínio público. O ldp-portable foi escolhido por ser o mais estável e conhecido entre as implementações de domínio público.

Para a implementação é necessário criar um ambiente MPLS/LDP. O ambiente consiste em computadores rodando linux com modificações MPLS aplicadas ao kernel e softwares zebra com modificações LDP para a troca dinâmica de rotas.

O trabalho consiste em montar o ambiente e fazer modificações no LDP de forma que os roteadores, neste caso os computadores, se autenticem conforme a solução proposta.

2 - DESENVOLVIMENTO DO TRABALHO

2.1 - REVISÃO BIBLIOGRÁFICA

O cronograma inicial do trabalho previa a realização de um aprofundamento teórico no assunto a ser desenvolvido. Para o desenvolvimento do tema foram necessários conhecimentos específicos nos protocolos MPLS e LDP. Além disso foi fundamental o conhecimento no protocolo TCP/IP, pois este foi o protocolo utilizado em todos os testes, além de ser usado pelo software zebra e o protocolo LDP.

Para a montagem do ambiente também necessitou-se compreender o funcionamento do sistema operacional linux e do software zebra, bem como de diversas ferramentas de apoio make, lilo, autoconf, automake, gcc entre outros.

Para a implementação do protótipo ainda foram necessários conhecimentos em criptografia assimétrica e criptografia unidirecional (Hash). Foram utilizados os algoritmos RSA e SHA-1, ambos inclusos nas bibliotecas do openssl, sendo assim necessário o conhecimento de utilização destes algoritmos.

Dessa forma, as páginas seguintes descrevem o que seria uma fundamentação teórica necessária para o desenvolvimento e entendimento do trabalho.

2.1.1 - MPLS

MPLS é uma tecnologia de rede que utiliza técnicas como engenharia de tráfego e comutação de pacotes de forma a amenizar problemas de redes IP tradicionais. Os principais problemas das redes IP e que motivaram o desenvolvimento de uma nova tecnologia de rede são:

- roteadores situados no núcleo ou periferia de *backbone* acabam sobrecarregados com um excessivo número de entradas nas tabelas de roteamento;
- as buscas nas tabelas de roteamento são demoradas pois é necessário um cálculo para encontrar o maior prefixo de endereço que coincide com o endereço de destino do pacote (*Longest Match*);
- o pacote é roteado individualmente por todos os roteadores do caminho entre a fonte e o destino, gerando atraso na propagação;

- o roteamento hop-by-hop leva, às vezes, a um desequilíbrio na rede gerando congestionamento em alguns roteadores enquanto outros roteadores ficam subutilizados;

Estes problemas impossibilitam a utilização de redes IP para alguns tipos de tráfego, em especial tráfego multimídia. Para solucionar o problema desenvolveu-se diversas tecnologias de comutação rápida, trocando-se o endereçamento IP de tamanho variável por rótulos pequenos e de tamanho fixo, chamados labels. Os roteadores, nestas tecnologias, tomam decisões baseados nas informações contidas nos labels e não mais no endereço de destino do pacote. As primeiras tecnologias desenvolvidas eram proprietárias e incapazes de interoperar. O MPLS é uma tecnologia não proprietária, documentada na RFC 3031, que resolve os diversos problemas citados acima.

O MPLS (Multiprotocol Label Switching) é um Framework definido pelo IETF (Internet Engineering Task Force) que proporciona designação, encaminhamento e comutação eficientes de fluxos de tráfego através da rede. É uma técnica de comutação de pacotes baseada em rótulos [Müller,2002].

Embora o interesse da comunidade que desenvolve o MPLS seja voltado para melhoria no tráfego IP, o MPLS pode ser aplicado a todos os protocolos da camada 3 (camada de rede) e pode ser implantado sobre diversas redes, entre elas redes ATM, Frame Relay, Ethernet e X.25.

Alguns conceitos são importantes para o entendimento do MPLS e do projeto, são eles:

LSP (Label Switch Path) – O LSP é um caminho virtual definido no domínio MPLS, muito similar aos caminhos virtuais em uma rede ATM (par VPI/VCI). O LSP pode ser criado por configuração manual ou por roteamento dinamicamente. Depois de ser criado o LSP o tráfego é encaminhado através dos roteadores, sem a necessidade de roteamento. Os LSP são unidirecionais.

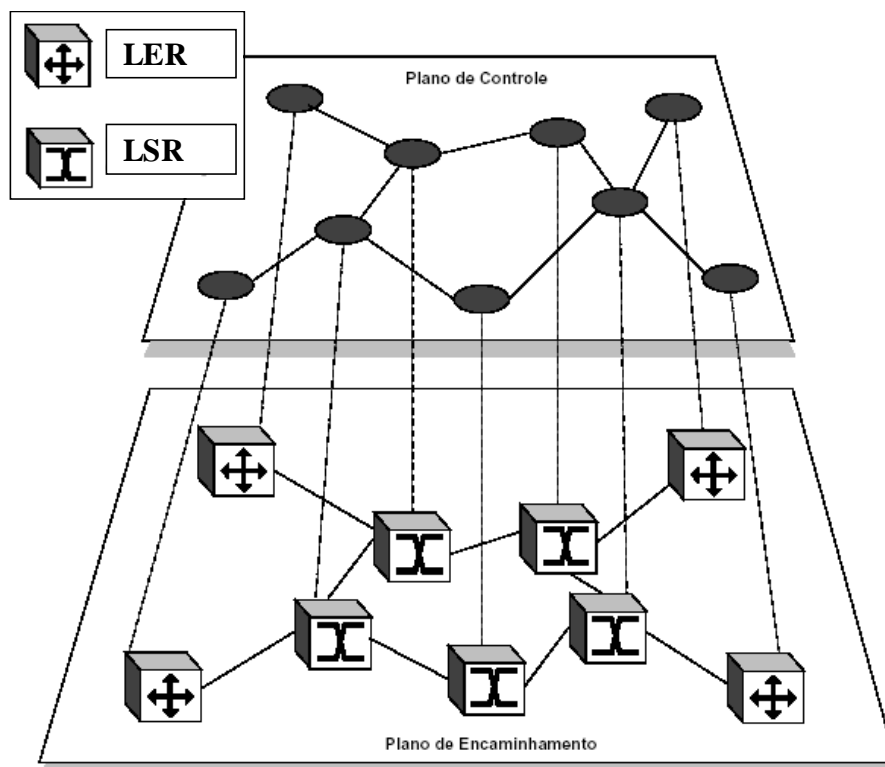
LSR (Label Switch Routers) – É o roteador em um domínio MPLS, tem a função de encaminhar o tráfego e opcionalmente pode colaborar na criação do LSP.

FEC (Forward Equivalence Class) – O agrupamento de todos os pacotes da camada 3 que serão encaminhados da mesma maneira é chamado FEC. Todos os pacotes do grupo usarão o mesmo caminho, ou LSP, e terão a

mesma prioridade no encaminhamento. Cada LSP está associado a uma Classe Equivalente de Encaminhamento, ou FEC. O roteador de ingresso do domínio MPLS tem a associação FEC/LSP sabendo, através do endereço da camada 3, como cada pacote deve ser encaminhado através da rede.

LER – (Label Edge Router) – Devido às especificidades de um LSR quando este está na entrada (ingresso) ou saída (egresso) de um domínio MPLS, o LSR é chamado LER. É importante frisar que o LSR é um LER apenas para as FECs vinculadas a(s) interface(s) de entrada/saída do domínio MPLS.

Um domínio MPLS pode ser dividido em dois planos funcionais: Plano de encaminhamento e plano de controle. O plano de encaminhamento é responsável por funções de sinalização, roteamento, conversão de endereços, policiamento de tráfego, dentre outras. A representação do plano de controle pode ser feita através de um grafo, já que é desta forma que os protocolos de roteamento enxergam a rede. O plano de encaminhamento é responsável pelo tráfego de dados, agregando as seguintes funções: encapsulamento, segmentação e remontagem, e rotulação de datagramas. Todo seu funcionamento é ditado pelo plano de controle. A figura 1 mostra o domínio



MPLS dividido em plano de controle e plano de encaminhamento.

Fig 1. Plano de controle e plano de encaminhamento

2.1.1.1 - Plano de encaminhamento

Ao entrar no domínio MPLS, o pacote recebe uma pequena estrutura denominada Shim Header que fica posicionada entre o cabeçalho de enlace e sua carga (pacote da camada 3) e que armazena um rótulo associado ao datagrama. Esta estrutura possui um rótulo de 20 bits, um campo experimental (EXP) de 3 bits, um campo TTL de 8 bits e um campo B de 1 bit cujo objetivo é indicar se o rótulo corresponde, ou não, ao último de uma pilha de rótulos, permitindo o encapsulamento de múltiplos rótulos.

O rótulo, também chamado de etiqueta ou label, é trocado por outro rótulo a cada LSR. É importante ressaltar que não há nenhum cálculo para esta troca, apenas a busca de um novo rótulo em uma tabela de rótulos residente no LSR, sendo assim muito mais rápido quando comparado à um ambiente de roteamento IP. Pelo fato do rótulo ter um tamanho fixo e ser pequeno, a procura pelo novo rótulo também é facilitada. A estrutura do shim-header é mostrada na figura 2. Os outros campos do shim-header não tem relevância para o projeto de autenticação.

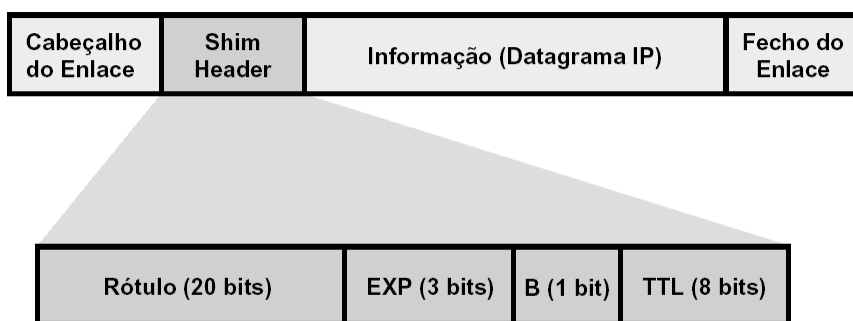


Figura 2 Estrutura do pacote MPLS.

O pacote MPLS atravessa o domínio sem alterações no campo de informação, onde está contido o pacote da camada 3 (IP, IPX, etc.). O cabeçalho do enlace e o fecho do enlace dependem da rede física existente entre os LSRs, sendo possível que o pacote MPLS atravessasse diferentes redes físicas tecnologicamente diferentes, como por exemplo ATM e Ethernet.

A figura 3 mostra como são trocados os rótulos em um pacote IP atravessando um domínio MPLS. Cada LSR consulta sua tabela de rótulos,

também chamada de LIB ou Label Information Base para saber qual o rótulo será o próximo.

Neste exemplo o LSR R1 inseriu o rótulo 50 e o envia para a porta 5 (interface 5). O LSR R1 atua como um LER de ingresso para este LSP. O LSR R2 consulta o rótulo 50 na LIB, troca o rótulo 50 pelo rótulo 20 e o envia para a interface 3. O LSR R4 consulta o rótulo 20 na LIB, não encontrando out label correspondente, então retira o shim-header e o envia para a interface 1. O LSR 4 atua como LER de egresso para o LSP.

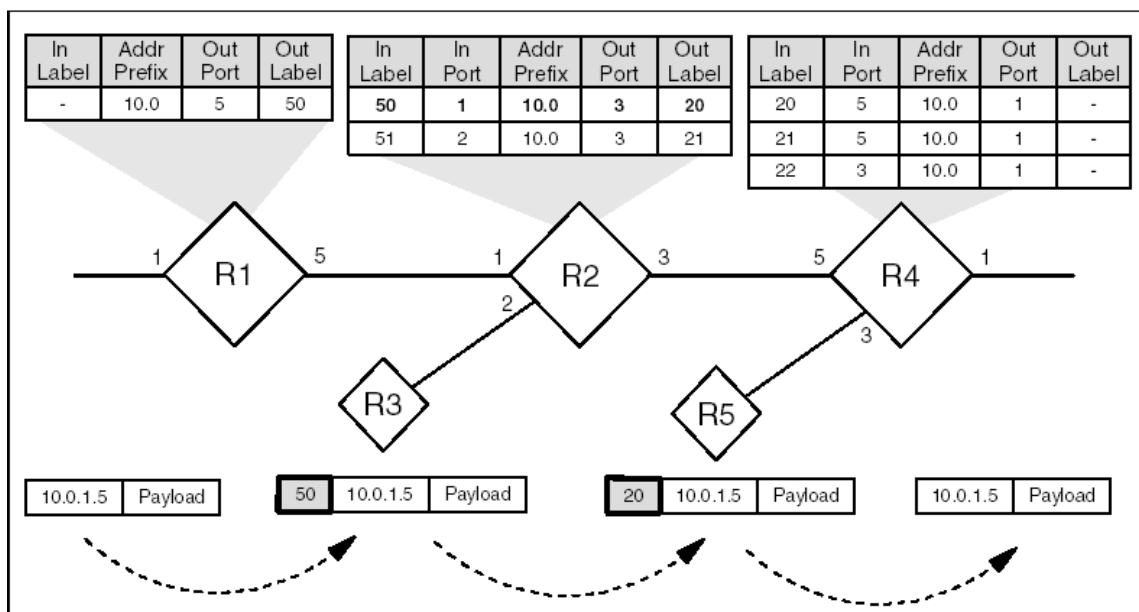


Fig. 3 – Pacote IP sendo transportado através de um domínio MPLS.

Ao sair do domínio MPLS o shim-header é retirado e o pacote volta a ficar apenas com os campos dos protocolos da camada 2 e 3. Como o shim-header fica entre os cabeçalhos da camada 2 e 3, o MPLS é considerado um protocolo da camada 2,5.

O MPLS prevê a possibilidade de um domínio MPLS se situar dentro de outro domínio MPLS. A figura 4 mostra um diagrama desta situação.

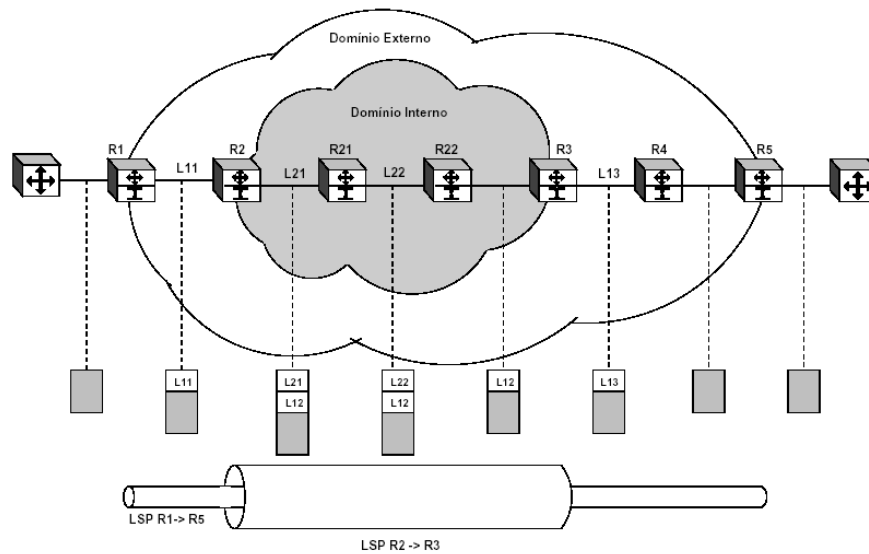


Fig 4. Domínios MPLS interno e externo.

Cada LER de ingresso de um domínio MPLS insere o seu shim-header, criando-se assim uma pilha de rótulos. O campo “B” do shim-header sinaliza a existência outro shim-header empilhado. Os LSRs trabalham apenas com o shim-header do topo da pilha. A figura 5 mostra a pilha de rótulos.

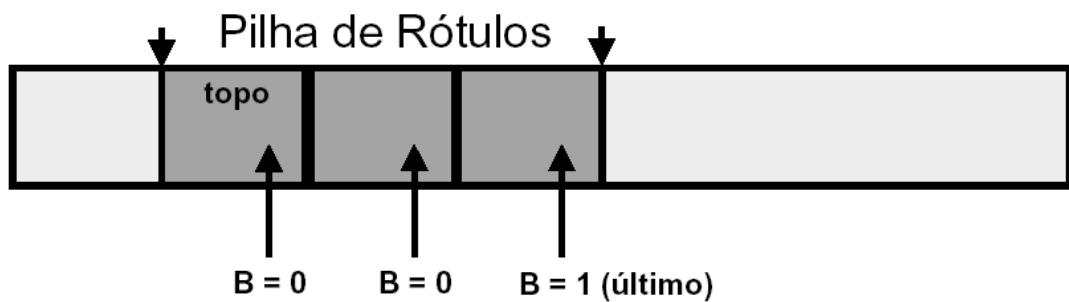


Fig 5 – Empilhamento de Rótulos

2.1.1.2 - Plano de controle

A construção de caminhos comutados em uma rede pode ser feita baseada na topologia da rede ou através do fluxo de dados. O MPLS utiliza-se da informação sobre a topologia da rede para a criação dos caminhos comutados, ou LSPs. Por utilizar informação sobre a topologia da rede, o MPLS é capaz de:

- 1 – Agregar tráfego para fluxos que convergem para um único ponto de egresso, diminuindo o número de LSPs necessários;
- 2 – Menor sobrecarga na sinalização. Os caminhos são criados baseados no roteamento da camada 3, que tem alterações mais lentas do que o fluxo de dados;
- 3 - Crescimento dependente do tamanho da rede. O número de LSPs tem relação ao tamanho da rede e não ao número de aplicações produtoras/consumidoras de fluxo que se comunicam através do domínio MPLS.
- 4 – Possibilidade de criação dos LSPs através de mecanismos de roteamento, inclusive mecanismos de gerência de rede. Pode-se assim criar políticas de engenharia de tráfego.

Como desvantagem, os LSPs continuam ativos, mesmo quando não existe tráfego de dados.

A arquitetura MPLS não impõe uma estratégia específica para a distribuição de rótulos, e conseqüentemente a criação de LSPs . Um domínio MPLS pode ter seus LSPs criados de forma manual, dinâmica ou ambos. Para a criação totalmente manual de um LSP é necessário configurar todos os LSRs onde este LSP irá passar, normalmente através de linha de comando. O método manual pode ser viável para domínios MPLS pequenos, porém para domínios maiores o método dinâmico é mais adequado.

2.1.1.2.1 – Distribuição de Rótulos

A distribuição de rótulos, mais corretamente relações rótulo-FEC, é necessária quando os LSRs não são configurados de forma manual.

Conceitos de *Downstream* e *Upstream* dizem respeito ao sentido do fluxo de pacotes em um LSP, na comunicação entre dois LSRs. Os pacotes sempre viajam de um LSR *upstream* (ou seja, o LSR anterior quanto ao sentido do fluxo da comunicação), para um LSR *downstream* (ou seja, o LSR posterior quanto ao sentido do fluxo da comunicação) [Müller,2002].

A distribuição de rótulos sempre é feita em sentido downstream, ou seja, os rótulos são gerados e distribuídos do LER de egresso para o LER de ingresso. Downstream pode ser traduzido como jusante e upstream pode ser traduzido como montante. A figura 6 mostra o sentido do fluxo de dados e o sentido da atribuição de rótulos.

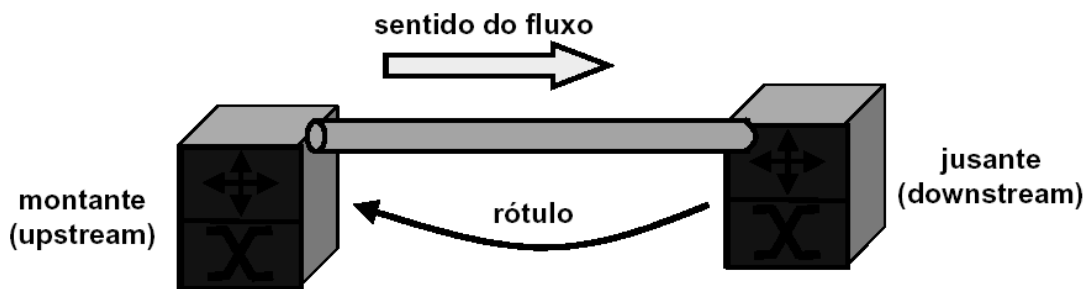


Figura 6 – Sentido do fluxo de dados e distribuição de rótulos

Um LSR pode requisitar explicitamente ao seu par, ou LSR adjacente, por um rótulo. Este método é conhecido como “Distribuição Sob Demanda”. Se o LSR distribui um rótulo, mesmo que outro LSR não tenha requisitado, o método é conhecido como “Distribuição Não Solicitada”.

A distribuição de associações rótulos, pode ter controle ordenado ou independente. No controle ordenado, o LSR distribui um rótulo somente se possui o rótulo. No controle independente, o LSR gera um rótulo para o LSR pedinte mesmo sem ter o rótulo para esta FEC.

Quando a estratégia de distribuição de rótulos for Sem Solicitação, o modo de retenção das associações pode ser Conservativo ou Liberal. No modo conservativo, o LSR distribui o rótulo somente se o rótulo conter o próximo hop para a FEC. Uma associação que não satisfaz esta condição é descartada no modo de retenção conservativo, e no modo liberal, ao contrário, é mantida. No modo liberal, a associação é mantida visando contemplar a eventualidade do LSR *upstream* se tornar o próximo *hop* no futuro (por exemplo, caso a topologia da rede se altere). O modo de retenção liberal faz sentido apenas em redes cuja dinâmica provoque alterações freqüentes das tabelas de roteamento [Müller,2002].

Em redes que utilizam identificadores de conexão, inclusive redes MPLS, os identificadores de conexão são recursos limitados. A distribuição de rótulos, deve minimizar o desperdício destes recursos, sendo a estratégia de Distribuição Sob Demanda, com Controle Ordenado a mais adequada. Esta forma de distribuição de rótulos é a mais utilizada em redes MPLS.

A tabela 1 apresenta um resumo em quadro comparativo das estratégias de distribuição de rótulos:

Estratégia de Distribuição	Controle	Modo de Retenção
Sob Demanda (<i>on demand distribution</i>)	Ordenado ou Independente	
Não Solicitada (<i>unsolicited distribution</i>)	Ordenado ou Independente	Conservativo ou Liberal

Tabela 1 - Resumo das Estratégias de Distribuição de Rótulos

Quatro protocolos atualmente suportam a criação de LSPs em um domínio MPLS. Alguns são protocolos de roteamento conhecidos e foram estendidos para suportar o MPLS e outros foram criados exclusivamente para o plano de controle do MPLS. São eles:

- BGP *Border Gateway Protocol*
- CR-LDP *Constraint-based Routed Label Distribution Protocol*
- LDP *Label Distribution Protocol*
- RSVP Resource Reservation Protocol

O protocolo LDP, também definido pelo IETF, é o mais utilizado, em domínios MPLS, destes quatro protocolos.

2.1.2 – LDP

LDP é o protocolo responsável pelo entendimento entre os LSRs. O protocolo LDP assegura a criação e manutenção dos LSPs, através da troca de mensagens LDP. O LDP contribui para o domínio MPLS com as seguintes funções:

- descobrindo outros LSRs conectados ao link;
- estabelecendo sessões entre dois LSR;
- distribuindo rótulos “a jusante” utilizando qualquer estratégia (sob demanda, sem solicitação, etc.);
- estabelecendo, mantendo e destruindo LSPs.

Uma extensão ao LDP, o protocolo CR-LDP (Constraint-based Routing LDP) permite ainda que os LSPs sejam estabelecidos através de roteamento com restrição (exemplo: roteamento explícito com QoS).

2.1.2.1 - Funcionamento do LDP

LDP associa a cada LSP criado uma classe (FEC – *Forwarding Equivalent Class*) para definição dos pacotes que serão mapeados naquele determinado LSP. Cada FEC é especificada como um prefixo de endereço de qualquer comprimento ou um endereço IP completo, os endereços são retirados da tabela de roteamento camada 3.

Parceiros LDP (LDP Pares ou LDP Peers), são os LSRs que utilizam LDP para trocar informação sobre rótulos e FECs. Para a troca de informações os LDP Pares estabelecem um Sessão LDP (Conexão LDP ou LDP Session). A comunicação é realizada através de troca de mensagens LDP. Para a identificação do LSR gerador da mensagem, utiliza-se os identificadores LDP, que são compostos pelo identificador do LSR, globalmente único, mais o Espaço de Rótulo (*Label Space*), que é definido dentro do escopo do LSR.

Existem quatro categorias de mensagens que são trocadas entre LSRs Pares:

- mensagens de descoberta: utilizadas para anunciar e detectar a presença de um LSR no domínio;
- mensagens de sessão: usadas para estabelecer, manter e terminar uma sessão entre dois parceiros LDP;
- mensagens de distribuição de rótulos: são usadas para criar, modificar e suprimir associações rótulo-FEC;
- mensagens de notificação: são usadas para prover informações de estado da rede e sinalizar erros.

As mensagens do LDP são transportadas utilizando o TCP, exceto mensagens LDP de descoberta.

O primeiro passo para a troca de informações entre LSRs é a criação de uma sessão LDP. O estabelecimento de uma sessão LDP ocorre da seguinte forma:

- Descoberta do Par LDP. Normalmente o LSR envia mensagens HELLO para o endereço de grupo multicast 224.0.0.2 (all-routers multicast group). O LDP envia suas mensagens de descoberta como pacotes UDP para a porta 646 (LDP *well-know discovery port*). O mecanismo de descoberta

utilizando multicast, conhecido como mecanismo de descoberta básico do LDP (Basic Discovery), é mostrado na figura 7. Caso os LSRs não sejam adjacentes, o LSR pode enviar mensagem HELLO para o endereço unicast do outro LSR. Neste caso as mensagens são conhecidas como “Target Hello” e caracteriza o mecanismo de descoberta estendido do LDP (Extended Discovery). O mecanismo estendido é mostrado na figura 8.

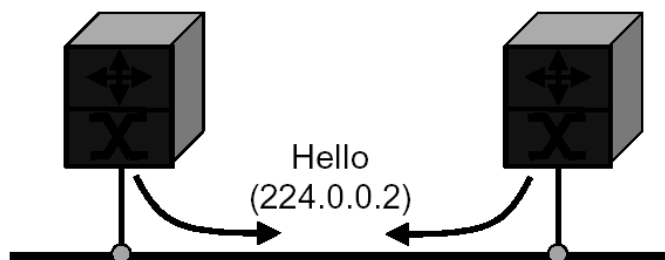


Figura 7 – Descoberta de LSR Par através de Multicast

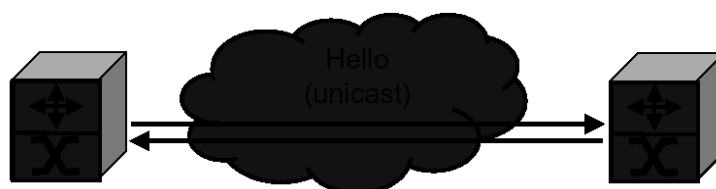


Figura 8 – Descoberta de LSR Par através de unicast

- Estabelecimento de Conexão da Camada de Transporte. É estabelecido uma conexão TCP entre os LSRs através da porta 646. No estabelecimento da sessão um LSR toma papel ativo e o outro passivo. O LSR com identificador (normalmente IP) toma o papel ativo da sessão. A figura 9 mostra este relacionamento entre os LSRs.

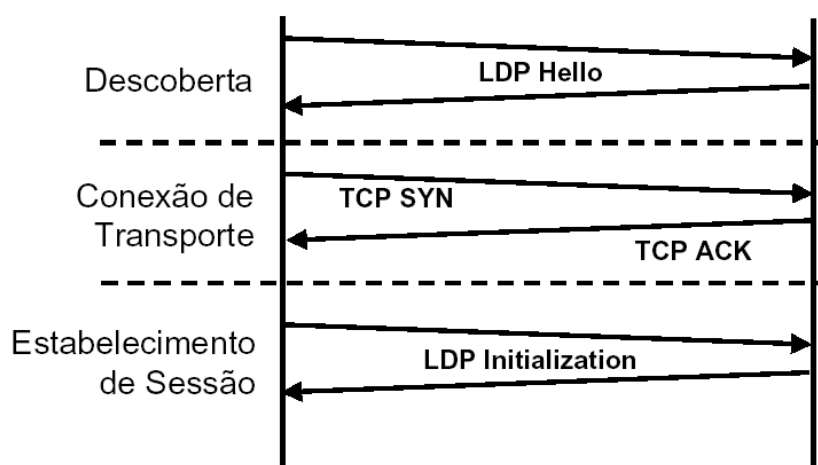


Figura 9 – Relacionamento entre LSRs Pares

- Inicialização da Sessão LDP. Após o estabelecimento de uma sessão TCP para o transporte de PDUs LDP, um dos lados envia uma mensagem tipo INITIALIZATION com informações quanto as características suportadas pelo LSR, como versão do protocolo, método de distribuição dos rótulos, tamanho máximo para PDUs e intervalo para envio de mensagens KEEP ALIVE. A figura 10 mostra a seqüência necessária para o estabelecimento de uma sessão LDP.

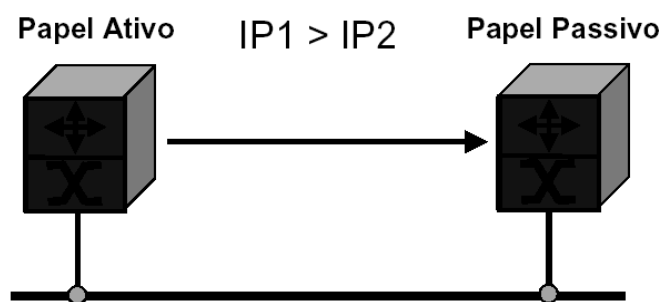


Figura 10 – Ordem para estabelecimento de uma sessão LDP

- Fase Ativa: Uma vez estabelecida a sessão LDP, os dois LSRs encontram-se na fase ativa para solicitação e distribuição de rótulos. Para a sessão se manter ativa é necessário manter a relação de vizinhança e manter a sessão LDP.

Para manter a relação de vizinhança, ou adjacência de HELLO, o LDP utiliza as mensagens regulares de HELLO (iniciadas pelo mecanismo de Descoberta do LDP) recebidas. O LSR mantém um contador para cada adjacência de HELLO (pode existir uma ou várias adjacências em uma sessão LDP entre dois pares) o qual é reiniciado a cada HELLO confirmado. Se o contador expirar, o LDP entende que o LSR par não deseja mais usar este espaço de rótulos ou que o par falhou. Assim o LDP exclui esta adjacência correspondente. Quando a última adjacência de HELLO de uma sessão LDP for excluída, o LSR termina a sessão LDP enviando uma mensagem LDP de NOTIFICATION e encerra a conexão de transporte (TCP).

A sessões LDP são mantidas através de mensagens periódicas do tipo KEEP_ALIVE. O intervalos entre as mensagens KEEP_ALIVE tem intervalos

regulares negociados pelos LSRs na fase de inicialização da sessão. O LSR mantém um contador KEEP_ALIVE para cada sessão que possui com um par específico. Se o contador expirar, o LSR conclui que a conexão está ruim ou que o par falhou. Então termina a sessão LDP encerrando a conexão de transporte (TCP). Um LSR pode escolher terminar um sessão LDP com um par a qualquer hora. Neste caso ele informa este fato ao par através de uma mensagem LDP de SHUTDOWN.

A figura 11 mostra como funciona a manutenção e encerramento de uma sessão LDP.

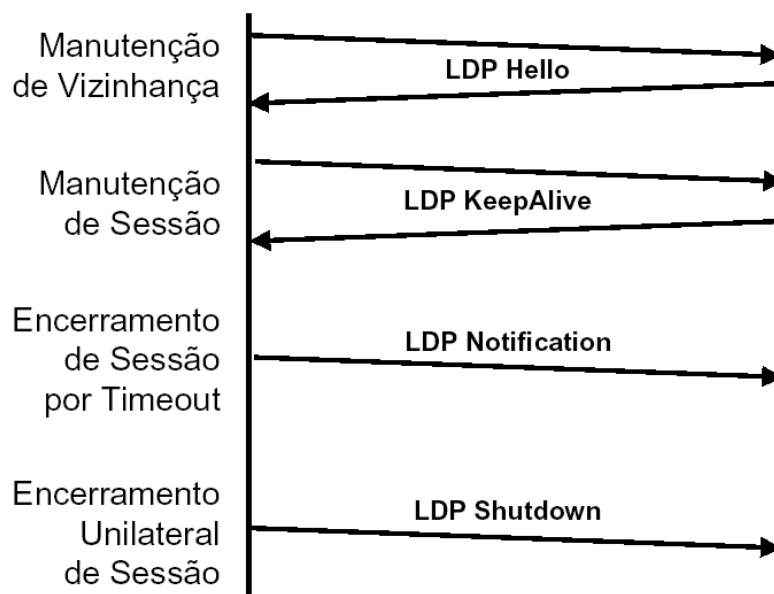


Figura 11 – Manutenção e encerramento de uma sessão LDP.

2.1.2.2 - PDUs LDP e TLVs

PDUs (*Protocol Data Units*) LDP definem o formato das pacotes LDP que contém um cabeçalho e um ou mais TLVs (*Type-Length-Value*), ou seja, tem o formato: CABEÇALHO LDP + TLV + [TLV, TLV, TLV, ...]. Os TLVs transportam as mensagens LDP. As mensagens LDP transportadas numa mesma PDU não necessitam ter qualquer tipo de relação entre si, ou seja, é possível serem transmitidos diversos tipos de TLV em uma única PDU. Por exemplo, uma PDU LDP pode transportar uma mensagem de LABEL REQUEST para um conjunto

de FECs e outra mensagem de LABEL MAPPING para outro conjunto de FECs. A Figura 12 mostra o formato do cabeçalho da PDU LDP.

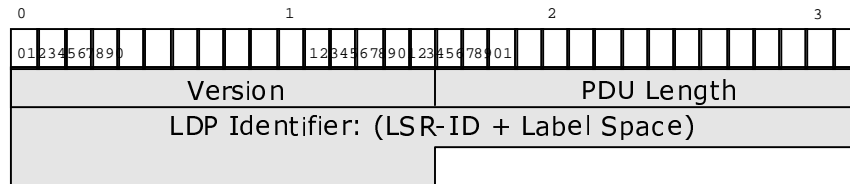


Figura 12 - Formato do Cabeçalho LDP de uma PDU LDP

Abaixo estão descritos os campos do cabeçalho de uma mensagem LDP:

Campo **Version**: versão do protocolo LDP em uso.

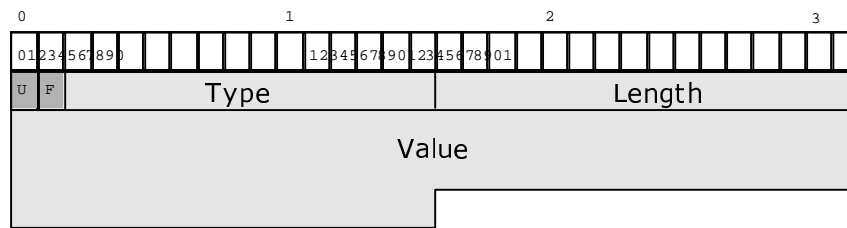
Campo **PDU Length**: Dois octetos especificam o tamanho total da PDU LDP em octetos, excluindo os campos “version” e “PDU Length”. O Tamanho padrão da PDU LDP é 4096 bytes, porém este valor é negociado no estabelecimento de uma sessão LDP entre dois pares.

O Campo **LDP Identifier** é um identificador único do espaço de rótulos de um LSR. É composto por 6 octetos, sendo que os primeiros 4 octetos identificam o LSR (LSR-ID) e os 2 octetos restantes identificam o espaço de rótulos (*label space*) em uso pelo LSR. O campo *LDP Identifier* possui a seguinte representação nominal: <LSR-ID>:<Label Space>, por exemplo "192.168.1.1:0"

LSR-ID identificador globalmente único do LSR, por exemplo o IP do LSR.

Espaço de Rótulos (*Label Space*): O espaço de rótulos pode ter dois significados de uso: por interface ou por plataforma. Quando usado por interface identifica unicamente cada interface do LSR e só pode ser usado entre LSRs diretamente conectados por enlace, por exemplo, um LSR com interfaces ATM que usa os VCIs como rótulos. Quando usado por plataforma, significa que as interfaces do LSR podem compartilhar o mesmo espaço de rótulos. Uma comunicação entre dois LSRs pode utilizar vários espaços de rótulo, por motivos diversos, nesse caso é criada uma sessão TCP/LDP exclusiva para cada espaço de rótulos em uso com o LSR par. [Müller, 2002].

O Formato da estrutura TLV é mostrada na figura 13:



13 - Formato de um TLV de uma PDU LDP

Abaixo está descrito o significado de cada campo de um TLV:

Campo **U-Bit**: *Unknow* TLV bit. Quando do recebimento de um TLV não reconhecido, o LSR deve ler o campo U-Bit. Se estiver em “0” toda mensagem LDP deve ser ignorada e deve ser retornada uma notificação para o emissor. Se estiver em “1” o TLV deve ser ignorado e o restante da mensagem processada.

Campo **F-Bit**: *Forward* TLV bit. Quando do recebimento de um TLV não reconhecido, o LSR deve ler o campo F-Bit. Se estiver em “0” o TLV desconhecido e toda mensagem LDP não são encaminhados adiante, ou seja, a mensagem é descartada. Se estiver em “1” o TLV deve ser ignorado e o restante da mensagem encaminhada para o próximo *hop*.

Campo **Type** (tipo): identifica como o campo “*value*” (*valor*) deve ser interpretado. Tipos de TLV definidos na especificação atual do LDP podem ser verificados em [Andersson, 2001].

Campo **Length** (tamanho): Especifica o tamanho do campo “*value*” em octetos.

Campo **Value** (*valor*): cadeia de octetos que codifica informações que devem ser interpretadas conforme especificado no campo “*Type*”. É importante salientar que o campo “*value*” pode conter outros TLVs, ou seja, os TLVs podem ser aninhados, o que é comum na maioria das mensagens LDP. [Müller, 2002].

2.1.2.3 - Gerência de Rótulos

Após ser criada a sessão LDP, os LSR estarão aptos para trocar mensagens para criação, distribuição e exclusão de associações Rótulo-FEC.

No modo de Distribuição Sob Demanda, Para criar uma nova associação, uma mensagem tipo LABEL REQUEST é enviada para o LSR posterior (downstream). A mensagem de LABEL REQUEST carrega a FEC para a qual se deseja associar um rótulo e opcionalmente, dois TLVs podem ser incluídos na mensagem:

- *hop count*: contém a contagem de número de hops, ou seja, o número de quantos LSRs propagaram a mensagem de requisição ou o comprimento do LSP até o presente LSR. Este TLV é importante para decrementar todo o TTL (*Time to Live*) no LSR de ingresso;
- *path vector*: vetor contendo a identificação de todos os LSRs que propagaram a mensagem de requisição. Esta informação é utilizada para a detecção de loops. Se um LSR recebe uma mensagem LABEL REQUEST e faz parte do path vector desta mensagem, então a mensagem percorreu um laço (loop).

O LSR aloca um rótulo para a FEC ao receber uma mensagem LABEL REQUEST. Se o LSR for de egresso para a FEC, o rótulo é prontamente retornado para o LSR requisitante. Caso contrário, e se o LSR opera no modo de controle ordenado, a requisição é propagada para o LSR que é o próximo hop para esta FEC. Quando esta requisição for respondida pelo próximo hop, o rótulo alocado é retornado para o LSR requisitante. No modo independente, o rótulo em geral é prontamente retornado, mesmo se o LSR não for o de egresso para a FEC. [Magalhães, 2001].

Para distribuir uma associação rótulo-FEC é empregada a mensagem LABEL MAPPING. Esta mensagem contém duas informações obrigatórias: a FEC e o rótulo atribuído. Se a estratégia de distribuição for Sob Demanda, a mensagem LABEL MAPPING é gerada em resposta a uma mensagem LABEL REQUEST. Caso a estratégia de distribuição for Sem Solicitação, um LSR pode gerar uma mensagem LABEL MAPPING sem ter recebido uma requisição, ou seja, sem ter recebido uma mensagem LABEL REQUEST.

Opcionalmente uma mensagem LABEL MAPPING pode conter mais três TLVs:

- *hop count* e *path vector*: com as mesmas funções nas mensagens LABEL REQUEST;
- *message ID*: usado para identificar o LSR que enviou o LABEL REQUEST;

Para excluir ou destruir uma associação rótulo-FEC existem duas mensagens LDP:

- LABEL WITHDRAW: utilizado para invalidar uma associação Rótulo-FEC atribuída pelo emissor e enviada via *upstream*. Esta mensagem é gerada quando o emissor não mais reconhece a FEC para a qual havia atribuído um rótulo.
- LABEL RELEASE: mensagem enviada a um LSR via *downstream* para informar que um mapeamento anteriormente requisitado pelo emissor não é mais necessário. Esta situação em geral está associada a mudanças no roteamento que provocam alterações no próximo *hop* para uma dada FEC. Ao receber esta mensagem, o LSR responde com uma mensagem tipo LABEL WITHDRAW, confirmando a invalidação da associação Rótulo-FEC.

LABEL WITHDRAW e LABEL RELEASE tem como efeito a alteração das associações Rótulo-FEC nas LIBs dos LSRs causando, para efeito prático, a destruição parcial ou total de LSPs.

A Figura 14 mostra um exemplo de troca de mensagens entre os LSP desde a detecção de LSRs no barramento, criação da sessão TCP e LDP, criação e destruição de LSPs.

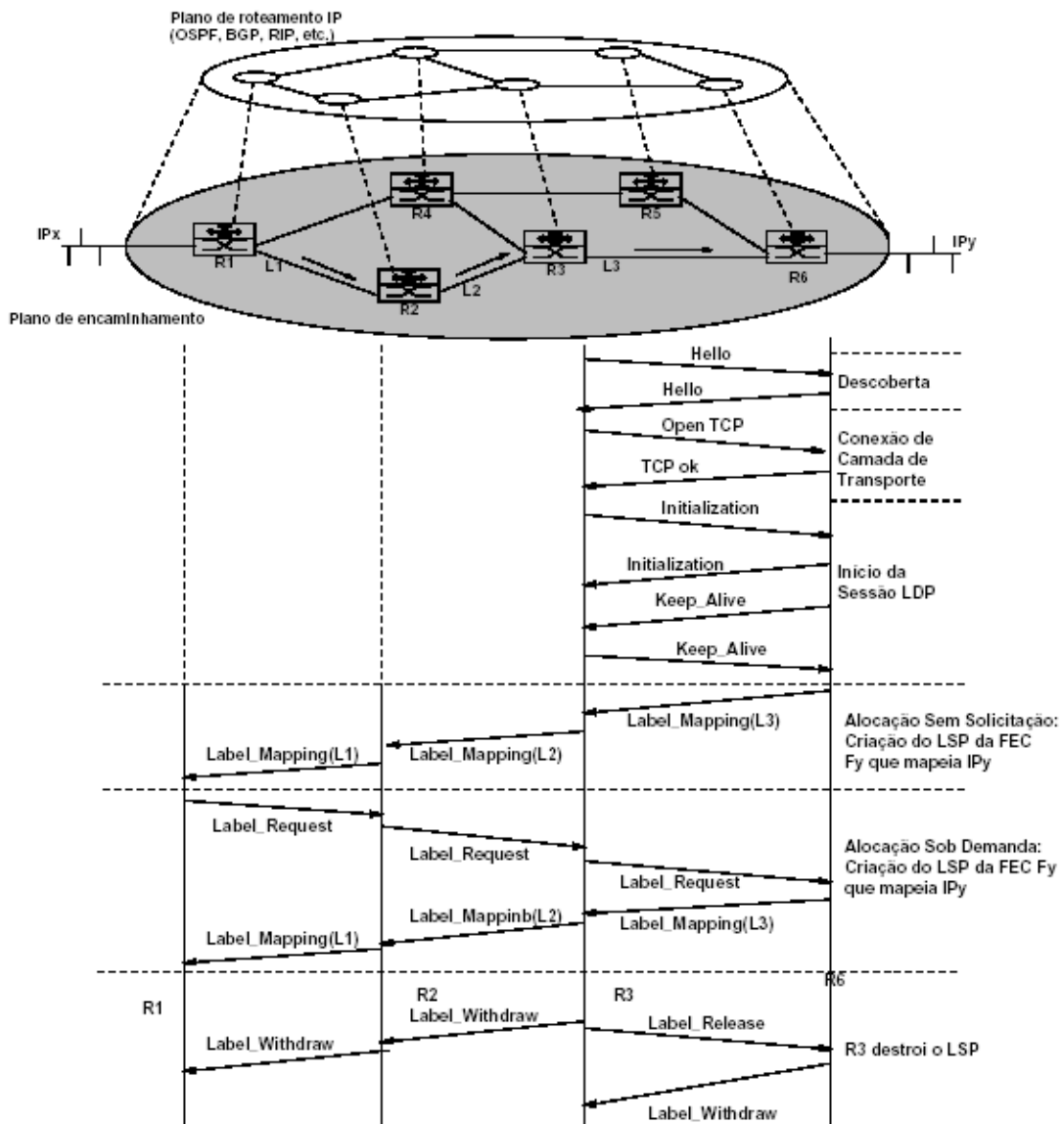


Figura 14 – Exemplo de mensagens LDP trocadas entre os LSRs

2.1.3 – TCP/IP

O TCP/IP é um conjunto de protocolos projetado para comunicação inter-redes. Os dois principais protocolos são o IP e o TCP, de onde advém a denominação.

O protocolo TCP/IP é o protocolo de interligação em redes mais utilizado na internet global e em redes corporativas e tornou-se um padrão de fato. O TCP/IP, que tem suas origens no início da década de 1970 e é desenvolvido principalmente por voluntários de universidades e empresas de todo o mundo.

O MPLS opera com diversas arquiteturas de rede física, de enlace e de rede. Apesar disso, o plano de controle está intimamente ligado ao TCP/IP. Por exemplo, as rotas para a criação dos LSP são retirados do roteamento IP, a sessão LDP opera sobre uma conexão TCP.

Para o entendimento do MPLS e do LDP é importante entender o funcionamento dos protocolos IP e TCP, em especial o protocolo IP.

2.1.3.1 - Protocolo IP

O protocolo IP é um protocolo que opera na camada de rede, ou camada 3 do modelo OSI/ISO (Open Systems Interconnection/ International Standards Organization). O protocolo IP é responsável pelo endereçamento e roteamento em uma rede TCP/IP.

2.1.3.1.1 - Endereçamento IP

Como o TCP/IP é um protocolo inter-rede, o IP provê um sistema de endereçamento para possibilitar um serviço de comunicação universal, logo quaisquer dois equipamentos conectados em uma rede TCP/IP devem poder se comunicar. Para isso, é que equipamento, ou host, possui um identificador único na rede. Esse identificador é conhecido como endereço IP ou endereço internet, e é constituído de um endereço inteiro de 32 bits. Um endereço IP define o identificador da rede na qual a máquina está conectada e também a identificação de um único host na rede. O endereço IP é um par (netid, hostid), onde netid identifica a rede, e hostid identifica um computador nessa rede. Os endereços podem estar incluídos em 5 classes distintas. As classes A, B e C são endereços unicast. A classe D são endereços de multicast e a classe E é reservada para uso futuro. As classes de endereçamento são mostradas na figura 15.

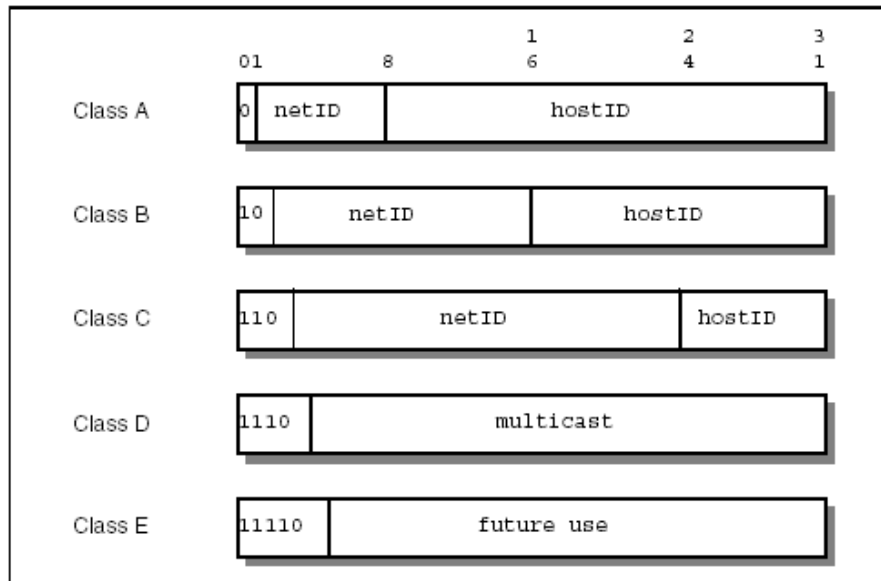


Figura 15 – Classes de endereçamento IP

Os endereços IP devem ser mapeados para endereços do protocolo da camada de enlace. Dependendo da rede física é abordada uma maneira diferente para esta conversão, como por exemplo o mapeamento direto (ex: proNet token ring) ou vinculação dinâmica (ex: Ethernet).

2.1.3.1.2 Roteamento IP

O roteamento em uma interligação em redes pode ser complicado, principalmente entre computadores que possuem várias conexões de rede físicas. Teoricamente, quando selecionasse o caminho, o software de roteamento examinaria itens como carga da rede, comprimento do datagrama ou o tipo de serviço especificado pelo datagrama. Entretanto, a maior parte dos softwares de roteamento da interligação em redes não é tão sofisticada assim e seleciona rotas tendo por base pressupostos fixos sobre caminhos mais curtos. [Comer, 1998].

O roteamento pode ser classificados em dois tipos: encaminhamento direto e encaminhamento indireto. No caso de encaminhamento direto, os hosts pertencem a uma mesma rede física e o datagrama IP pode ser transmitido diretamente. Quando o destino não se encontra na mesma rede física, ocorre o roteamento indireto, sendo necessário que o datagrama IP seja transmitido através de um ou mais roteadores.

Os roteadores de uma interligação em redes TCP/IP formam uma estrutura cooperativa e interconectada. Os datagramas passam de roteador a roteador até acessarem um que possa entregar o datagrama diretamente. [Comer, 1998]. A dificuldade no roteamento está em como cada roteador ou host sabe para onde devem ser enviado cada datagrama até que o destino seja alcançado.

A solução mais simples, e bastante utilizada, é cada host ou roteador possuir uma tabela de roteamento IP contendo informações sobre possíveis destinos e como acessá-los. Sempre que um host ou roteador precise transmitir um datagrama ele deve consultar esta tabela. Armazenar informações sobre todos os hosts ou redes em uma tabela é impraticável e ineficiente, na prática roteadores normalmente armazenam informações sobre redes conectadas diretamente ao próximo roteador. Desta forma os roteadores não conhecem o caminho completo percorrido por um datagrama.

Roteadores e hosts também podem armazenar em suas tabelas de roteamento IP informações sobre alguma rede ou host específico e podem ter uma entrada na tabela para uma “rota default”, por onde será encaminhado todo o resto do tráfego.

Com o tamanho e as constantes mudanças em redes TCP/IP, existe a necessidade da propagação automática de rotas. Conexões falham e são substituídas depois. As interligações em redes podem tornar-se sobrecarregadas em determinado momento e subutilizadas logo em seguida. A finalidade dos mecanismos de propagação de rotas não é meramente encontrar um conjunto de rotas, mas sim atualizar permanentemente essas informações. Os seres humanos simplesmente não podem responder às mudanças com a velocidade necessária; para tanto, devem ser usados os programas de computador. Dessa forma, quando pensamos em propagação de rotas, é importante considerar o comportamento dinâmico dos protocolos e dos algoritmos. [Comer, 1998].

Para propagação de rotas de forma dinâmica diversos protocolos e algoritmos foram implementados. São eles os protocolos RIP, OSPF, BGP, IS-IS, etc. O LDP cria os LSPs baseado nas tabelas de roteamento IP. Sendo estas rotas criadas de forma estática ou dinâmica.

2.1.3.2 - TCP

O TCP (Transmission Control Protocol), ao lado do UDP (User Datagram Protocol), são os dois protocolos da camada de transporte, ou camada 4, do TCP/IP. O TCP oferece um serviço de conexão fim a fim confiável. O protocolo TCP resolve problemas de pacotes perdidos ou danificados que ocorrem normalmente quando o hardware de rede falha ou quando as redes tornam-se muito sobrecarregadas.

O TCP sozinho, não garante integridade, autenticação ou confidencialidade e está sujeito a diversos ataques conhecidos e documentados.

O LDP utiliza o TCP para o estabelecimento da sessão LDP.

2.1.4 – LINUX

O Linux é um sistema operacional multiusuário e multitarefa, composto de um kernel (ou núcleo), programas e aplicações. O kernel é a principal parte do Linux, e é quem controla o acesso ao sistema de arquivos e suas operações, aloca memória, executa programas, gerência recursos entre os programas de usuário, entre outras coisas.

O linux tem é distribuído sob licença GPL, o que permite alterações no sistema por qualquer pessoa com conhecimentos necessários. Isto proporcionou ao linux, além de outras coisas, uma grande modularidade. O kernel também pode ser compilado com modificações ou novas funcionalidades.

No linux, a interligação em redes é controlada pelo próprio kernel. São responsabilidades do kernel: receber, enviar, fragmentar, defragmentar pacotes e também repassar pacotes entre interfaces e/ou processos.

O kernel oficial do linux possui código TCP/IP e Ethernet, ambos utilizados para a criação do ambiente de testes, porém não possui nenhum suporte a tecnologia MPLS.

O Linux cria um diretório virtual com informações sobre variáveis do sistema, estado de dispositivos, processos, etc. Este diretório chama-se “/proc” e é bastante útil para verificação de estado do sistema.

A versão do kernel utilizada no projeto foi a 2.4.19, a mais atual na momento da criação do ambiente e recomendada nas listas de discussão do projeto MPLS-LINUX. O endereço onde o kernel oficial do linux pode ser baixado é <http://www.kernel.org>

2.1.5 – MPLS-LINUX

MPLS-LINUX, ou MPLS for Linux, é um projeto de código aberto que implementa o protocolo MPLS na plataforma linux e é hospedado no endereço <http://sourceforge.net/projects/mpls-linux/> estando vinculado ao grupo "source forge" (<http://sourceforge.net>). O principal desenvolvedor do projeto é James Leu.

O MPLS-LINUX modifica o kernel do linux, acrescentando funcionalidade MPLS. É implementado o plano de controle do MPLS, além de criar um subdiretório chamado mpls no diretório /proc. No diretório /proc/mpls é possível verificar o espaço de labels, as associações rótulo-FEC, etc. O MPLS-LINUX também possui uma ferramenta para controle do MPLS de forma manual, chamada mplsadm.

A figura 16 mostra as alterações do fluxo de pacotes no kernel do linux para o processamento de pacotes do tipo MPLS:

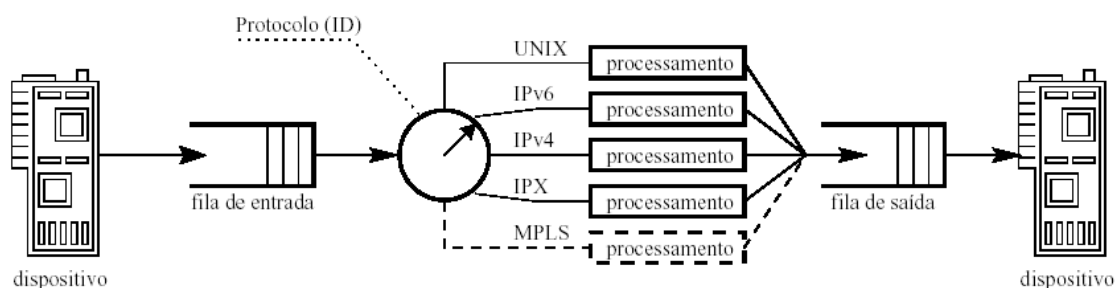


Figura 16 – Alterações no kernel do linux para processamento de pacotes MPLS

2.1.6 – ZEBRA

O projeto ZEBRA, hospedado em <http://www.zebra.org>, tem como objetivo implementar protocolos de roteamento IP dinâmicos, fornecendo uma interface de configuração e operação semelhante para todos os protocolos, a qual está baseada na interface de comandos dos roteadores CISCO.

Atualmente o projeto ZEBRA implementa os seguintes protocolos de roteamento dinâmico: BGP, OSPF, IS-IS, RIP e atualmente protocolo LDP do MPLS.

O zebra é bastante útil para que um computador que tem funções de roteador TCP/IP tenha uma interface para gerenciamento muito parecido com a interface de controle de um roteador CISCO. Desta forma, o administrador da rede tem uma interface homogênea para o gerenciamento de toda a rede.

Após instalado e configurado, ele abre uma porta TCP para aceitar conexões “telnet” para aceitar administração via linha de comando. Tipicamente, executa-se telnet para a porta do zebra e entra-se com usuário e senha para monitorar o roteador. Caso necessite-se alterar alguma configuração é necessário ainda entrar com o comando “enable” (modo privilegiado) e colocar a senha do administrador.

O zebra normalmente inicia uma instância para cada protocolo específico (ex: RIP, OSPF). Por exemplo, se o computador com Zebra estiver configurado para rodar OSPF, pode-se iniciar um daemon de OSPF que ficará escutando em uma porta específica. O funcionamento é idêntico ao do Zebra, podendo se alterar parâmetros apenas do OSPF.

2.1.7 – LDP-PORTABLE

O LDP-PORTABLE é um módulo do projeto MPLS-LINUX. O LDP-PORTABLE é escrito para adicionar funcionalidade LDP para o software Zebra, ou seja o LDP-PORTABLE não funciona de forma separada.

Para o funcionamento do LDP-PORTABLE é necessário executar atualizações no código fonte do Zebra, compilar o Zebra com estas alterações e rodar o zebra configurado para iniciar o módulo “mplsd”.

O módulo mplsd inicia um daemon que permite alterações no LDP via linha de comando. Para isso aguarda conexões em uma porta TCP específica.

2.1.8 – Criptografia Assimétrica

Criptografia assimétrica é a criptografia que utiliza o esquema de chaves públicas e privadas para garantir a segurança. A chave pública (public key) pode ser conhecida por qualquer um, enquanto a chave privada (private

key) é conhecida apenas pelo dono da chave. A chave pública não pode ser determinada através da chave privada e a chave privada não pode ser determinada através da chave pública. Um texto plano (cleartext) cifrado com a chave privada somente pode ser decifrado através da chave pública correspondente. Supondo que Alice queira enviar uma mensagem confidencial para Bob, ela pode cifrar a mensagem com a chave pública de Bob de forma que somente Bob poderá decifrar a mensagem. O texto cifrado (ciphertext) poderá ser enviado por um canal de comunicação não confiável.

A criptografia assimétrica é muito mais lenta que a criptografia simétrica, porém possui características únicas. É possível por exemplo autenticar uma mensagem cifrando a mensagem com a chave privada. Como todos podem ter acesso a chave pública e decifrar a mensagem, a privacidade (ou confidencialidade) não é garantida. O receptor decifrando a mensagem com a chave pública do transmissor pode garantir que a mensagem é realmente do transmissor pois mais ninguém tem acesso a chave privada dele. A figura 17 mostra um exemplo de uso de criptografia assimétrica com autenticação e confidencialidade.

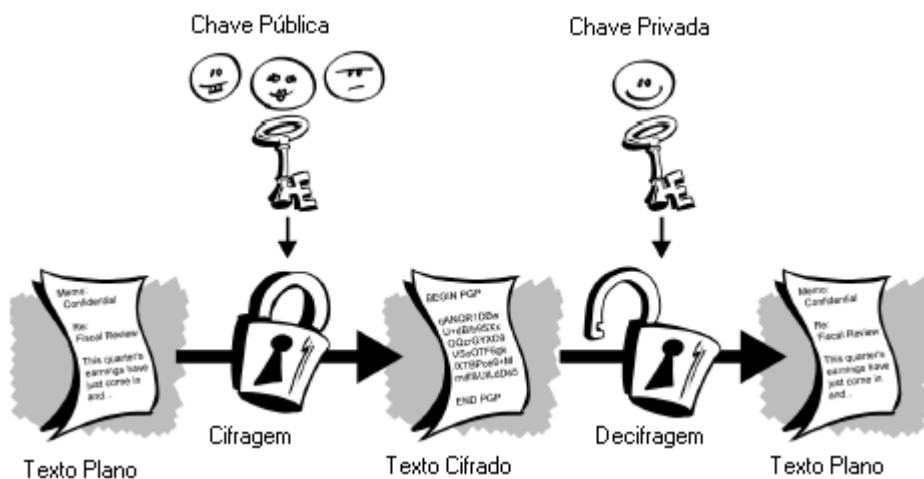


Figura 17 – Exemplo de criptografia assimétrica com confidencialidade

2.1.8.1 - RSA

RSA é o mais popular algoritmo de chave assimétrica e de fato um padrão mundial. O nome RSA advém do nome (sobrenome) de seus três inventores: Ron Rivest, Adi Shamir e Leonard Adleman. A segurança do RSA é baseada na dificuldade de fatorar números muito grandes. As chaves públicas

e privadas são funções de dois números primos muito grandes (200 dígitos ou mais). O RSA tem resistido durante muitos anos a extensivos ataques. Com o aumento do poder computacional, pode-se aumentar a segurança do RSA com o aumento do tamanho das chaves.

2.1.9 – Funções Hash

As funções hash (também chamada de criptografia unidirecional, criptografia de mão única, funções resumo ou message digest) são essenciais para a criptografia. A função hash é uma função que pega uma entrada de tamanho variável e produz uma saída tamanho fixo (o valor de hash). Se o valor de hash de duas mensagens são iguais, é improvável que as duas mensagens não sejam idênticas. A função de hash deve ser fácil de computar e impossível de ser revertida. Em uma boa função de hash também deve ser difícil de acontecer uma colisão, ou seja, para duas entradas diferentes o valor de hash ser o mesmo.

Uma função hash pode ter uma chave como um segundo parâmetro de entrada. Neste caso o valor de hash dependerá tanto da mensagem como da chave. Este método é chamado de message authentication code (MAC) e é mostrado na figura 18.

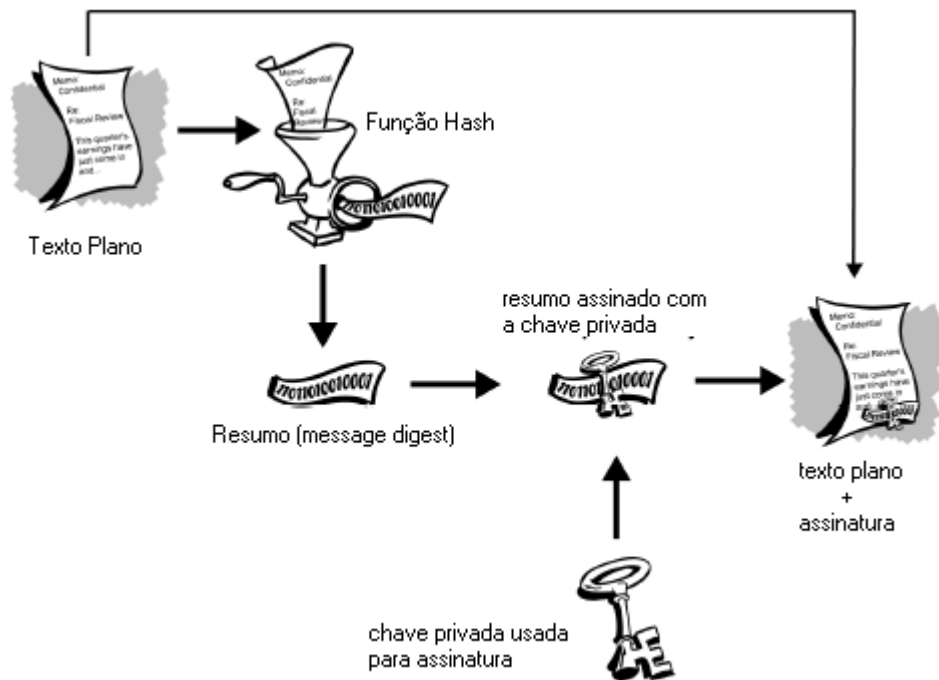


Figura 18 – Geração do message authentication code (MAC)

Tanto chaves simétricas como chaves assimétricas podem ser usadas para gerar o message authentication code.

Cifrando o valor de hash com a chave privada se obtém a assinatura digital (digital signature). A assinatura digital é um tipo especial de MAC. Usando assinatura digital ao invés de cifrar toda a mensagem com a chave privada ganha-se em performance, além de separar a autenticação da mensagem.

As funções de hash mais utilizadas são o MD5 e o Secure Hash Algorithm 1 (SHA-1). MD5 foi projetado por Ron Rivest (co-inventor do RSA). SHA-1 é baseado no MD5 e foi projetado pelo National Institute of Standards and Technology (NIST) e pelo National Security Agency (NSA) para uso com o DSS (Digital Signature Standard). MD5 produz 128 bits de hash, enquanto SHA-1 produz 160 bits de hash. O SHA-1 é uma função hash mais segura que o MD5 e é também a função utilizada no projeto.

2.2 – PROBLEMAS DE SEGURANÇA DO LDP

Diversas são as vulnerabilidades existentes no LDP. Este capítulo explica as vulnerabilidades que podem ser solucionadas com a utilização da autenticação proposta. Informações sobre outras vulnerabilidades do LDP podem ser encontrados em [Müller, 2002].

O LDP não define mecanismos próprios, num escopo por pacote, para autenticação, integridade e proteção contra ataques de repetição. Apesar de não possuir mecanismos próprios, o LDP pode utilizar a autenticação TCP baseada em MD5 (descrita na RFC2385), visto que as sessões LDP utilizam conexões TCP.

De modo a viabilizar uma autenticação entre dois LSRs não-adjacentes, onde não existe um sessão TCP/LDP fim a fim entre as pontas, por exemplo durante o estabelecimento do primeiro LSP entre eles, é necessário que sejam definidos mecanismos próprios ao protocolo LDP que possibilitem que este por si próprio possa fornecer algum mecanismo de autenticação [Müller, 2002].

Sem a segurança garantida pela autenticação, os pacotes de sessão LDP ficam vulneráveis a diversos ataques. Estando as sessões LDP

vulneráveis, os dados que atravessam o domínio MPLS também ficam vulneráveis. Os principais ataques são:

- um intruso pode tentar modificar os pacotes (de sessão ou de dados).
- um intruso pode tentar sequestrar uma conexão LDP.
- um intruso pode disparar um ataque de negação de serviço (*denial of service*) terminando conexões LDP, por exemplo enviando TCP *resets* (que determinam que a conexão deve ser reiniciada).
- um intruso pode tentar corromper a negociação entre dois pares LDP, de forma a ganhar acesso a chave compartilhada.

O LDP transporta mensagens de roteamento originados dos protocolos de distribuição de rotas IP (BGP, OSPF, etc). Como as mensagens de distribuição de rotas IP não são confidenciais, não há motivo para as mensagens LDP serem confidenciais.

Na especificação atual do protocolo LDP (RFC 3036), o mecanismo de autenticação TCP baseado em MD5 definido, provê autenticação mútua entre o par local e o par adjacente para o tráfego de dados e sessão TCP. No entanto a sessão LDP não é protegida, pois não garante autenticação, integridade e proteção contra ataques de repetição.

Outro problema é o fato do mecanismo atual não ser aplicável à comunicação entre LSRs não adjacentes, pois depende de uma conexão TCP estabelecida.

2.3 – AUTENTICAÇÃO FIM A FIM PARA O LDP

Este capítulo apresenta a proposta de autenticação fim a fim para o LDP, proposta por [Müller, 2002].

A proposta de autenticação modifica os procedimentos utilizados para a criação de um LSP.

Para o entendimento dos novos procedimentos, toma-se como exemplo uma rede MPLS onde LerA e LerB são respectivamente, ingresso e egresso do LSP e são LSRs não-adjacentes entre si e Lsr1 e Lsr2, Lsr2 e Lsr3, LerA e Lsr1, Lsr3 e LerB, são LSRs adjacentes. O exemplo é mostrado na figura 19.

Considera-se que:

- a) o LDP está configurado para operar no Modo de Distribuição Sob Demanda (isso significa que um LSR apenas vai distribuir um rótulo ao seu par LDP se este solicitar esse procedimento) e no Modo de Controle Ordenado em todos os LSRs do cenário;
- b) no ambiente está sendo executado um protocolo de roteamento IP dinâmico (OSPF) e através das informações de roteamento IP o LERA conhece um prefixo de endereços IP (10.1.0.0/8) que está "atrás" do LERB.
- c) o LERA deseja estabelecer um LSP para o prefixo de endereços IP 10.1.0.0/8, ou seja, par a FEC 10.1.0.0/8, situada atrás do LERB. Dessa forma o LERA será o LSR de Ingresso e o LERB será o LSR de Egresso para este LSP.

Quando o LDP é iniciado, os LSRs adjacentes iniciam o estabelecimento das sessões LDP entre si pelo envio de mensagens LDP HELLO ao endereço de multicast, através de pacotes UDP na porta 646. Isto significa que em duas etapas serão estabelecidas primeiro conexões TCP e em seguida sessões LDP entre os LSRs adjacentes, fazendo com que se tornem pares LDP e entrem na "fase ativa do LDP" onde poderão realizar operações com rótulos MPLS.

Após o estabelecimento das sessões LDP entre os LSRs adjacentes, o plano de controle do exemplo fica conforme a figura 20.

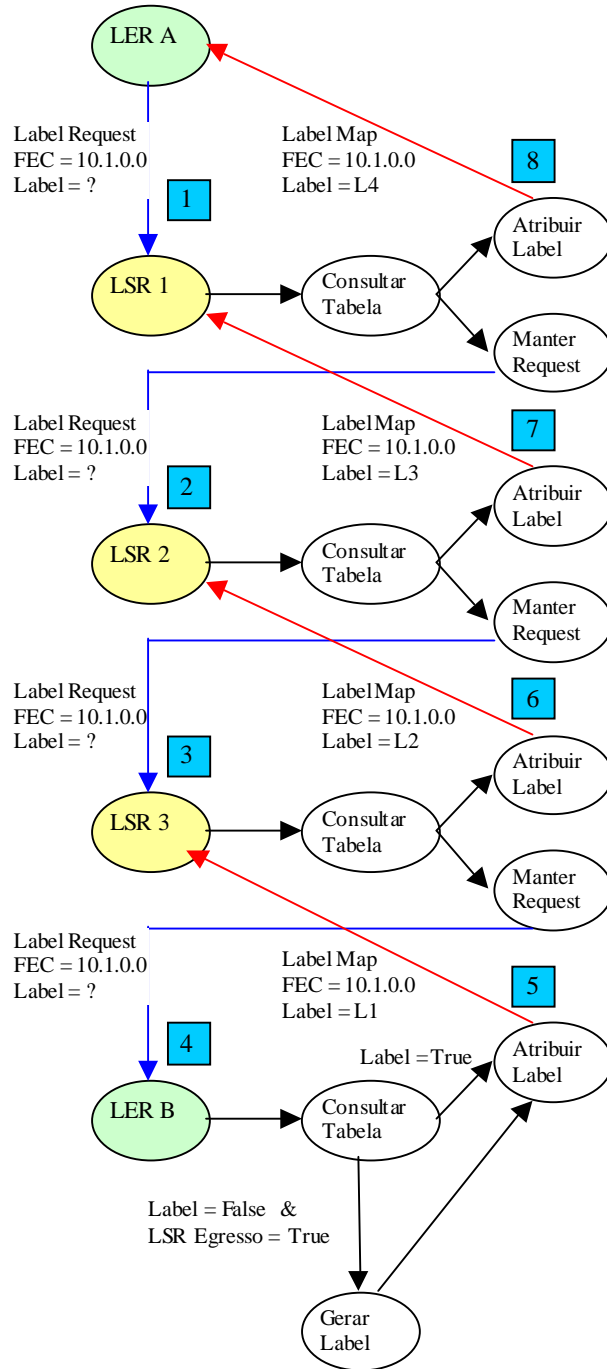
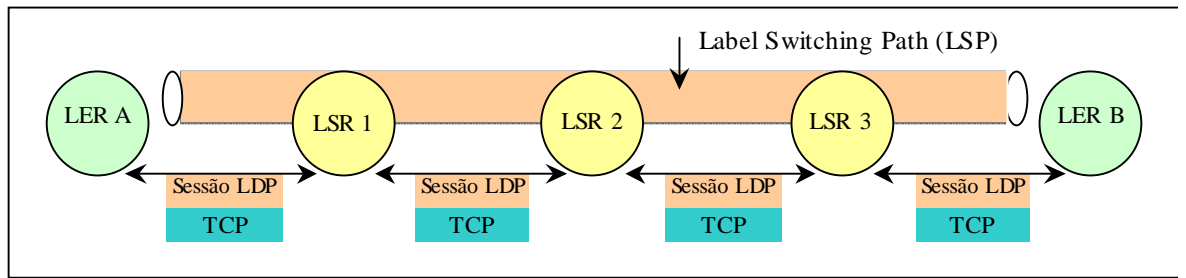


Figura 19 - Mensagens trocadas no estabelecimento de um LSP

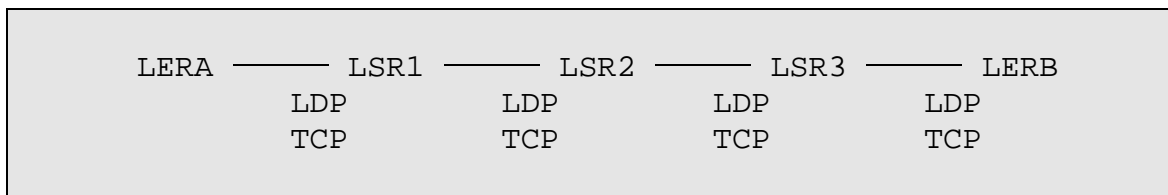


Figura 20. Plano de controle do LDP após o estabelecimento das sessões TCP/LDP

Com as sessões LDP estabelecidas, o LERA pode iniciar a solicitação de estabelecimento do LSP para a FEC 10.1.0.0/8. Através das informações da sua tabela de roteamento, o LERA sabe que o LSR adjacente LSR1 é o próximo roteador (*next hop*) no caminho para este prefixo de endereços IP. O procedimento executado é o seguinte:

- a) No passo 1, o LER envia uma mensagem LDP LABEL REQUEST para o LSR1 solicitando um rótulo para a FEC 10.1.0.0/8;
- b) No passo 2, O LSR1 realiza uma procura em sua tabela de rótulos local (Label Information Base - LIB) e percebe que não possui um rótulo para esta FEC. O LSR1 então altera o status da requisição recebida para "pendente", e baseado nesta requisição codifica e envia uma nova mensagem LDP LABEL REQUEST para o seu par adjacente LSR2, solicitando um rótulo para a FEC 10.1.0.0/8 (sobre a sessão TCP que inicia em LSR1 e termina em LSR2). O mesmo procedimento ocorre entre o LSR2 e o LSR3, e finalmente entre o LSR3 e o LERB, nos passos 3 e 4.
- c) No passo 5, O LERB reconhece a FEC 10.1.0.0/8 e percebe que é o LSR de egresso para o LSP em relação a esta FEC. Então o LERB gera um rótulo L1 para a FEC 10.1.0.0/8 e envia uma mensagem LDP LABEL MAPPING para o LSR3, informando este rótulo.
- d) No passo 6, o LSR3 insere um rótulo recebido em sua LIB e percebe que possui uma requisição pendente para a FEC 10.1.0.0/8 originada por LSR2. Então gera e envia um rótulo L2 para o LSR2 através de uma mensagem LDP LABEL MAPPING.
- e) No passo 7, O LSR2 insere o rótulo em sua LIB e percebe que possui uma requisição pendente para a FEC AR originada por LSR1. Então gera e envia um rótulo L3 para o LSR1 em uma mensagem LDP LABEL MAPPING (sobre a sessão TCP/LDP entre o LSR2 e o LSR1). A mesma coisa ocorre do LSR1 ao LERA (rótulo L4) no passo 8.

f) Quando o LERA recebe a mensagem LDP LABEL MAPPING com a FEC AR do LSR1 (via a sessão TCP/LDP entre o LERA e o LSR1), ele insere o rótulo recebido em sua LIB, percebe que é o LSR de Ingresso para a FEC 10.1.0.0/8 e neste ponto o LSP é estabelecido entre o LERA e o LERB. A partir deste ponto a FEC 10.1.0.0/8 está mapeada para este LSP e todos pacotes com destino a endereços do prefixo 10.1.0.0/8 serão roteados pelo LSP através do MPLS.

Neste ponto, no plano de encaminhamento (*forwarding plane*) temos a visão da pilha de protocolo mostrada na figura 20.

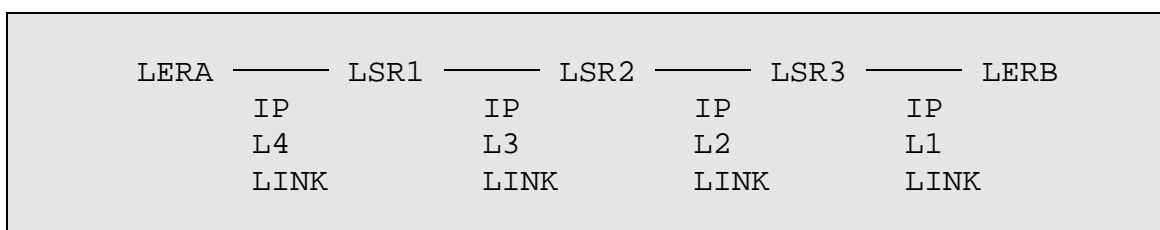


Fig 20 -Visão do Plano de Encaminhamento do LDP após o estabelecimento do LSP entre LERA e o LERB

O que a atual forma de autenticação definida para o LDP na RFC 3036, baseada em TCP/MD5 nos oferece é que, por exemplo, o LERB pode autenticar o rótulo L1 em relação ao seu par adjacente LSR3 (0), ou seja, apenas permite autenticar um LSR adjacente, pois entre ambos existe um conexão TCP direta. Não permite autenticar o LERA, por exemplo, já que o mesmo é não-adjacente em relação ao LERB.

Como entre o LERA e o LERB, no momento do estabelecimento do primeiro LSP, não existe uma sessão TCP/LDP fim a fim estabelecida, nenhuma forma de autenticação que funciona sobre o TCP pode ser empregada, como por exemplo, SSL, TLS, IPSec e outros.

2.3.1 - Proposta de Autenticação Fim a Fim para o LDP

A proposta, defendida por [Müller], prove uma solução para autenticar, num escopo fim a fim, o estabelecimento de um novo LSP entre um LSR de Ingresso e seu respectivo LSR de Egresso.

A solução foi planejada para contextos onde LSPs atravessam domínios de trânsito (ambientes multi-domínios não confiáveis) e os LSRs das bordas desejam se autenticar mutuamente. Na grande maioria dos casos os LSRs de ingresso e egresso de um LSP são LSRs não-adjacentes mas não é uma regra [Müller].

A autenticação definida na RFC 3036 só se aplica entre LSRs adjacentes, pois como é uma extensão do protocolo TCP (da camada de transporte), necessita de uma conexão TCP estabelecida entre os LSRs que vão se autenticar. Sendo assim este mecanismo não trata situações onde as extremidades de um LSP, ou seja, LSRs não-adjacentes que não possuem uma conexão TCP estabelecida diretamente entre si, pretendem se autenticar mutuamente, fim a fim, durante o estabelecimento de um LSP.

Uma comunicação entre LSRs não-adjacentes tem as seguintes características básicas:

- a) no momento do estabelecimento do primeiro LSP, não existe uma conexão TCP direta, fim a fim, entre os dois LSRs não-adjacentes que desejam estabelecer o LSP. O que existe são várias sessões TCP/LDP entre os LSR adjacentes do caminho;
- b) as mensagens LDP são encaminhadas ao primeiro LSR adjacente no caminho, deste para próximo, através das sessões TCP/LDP que existem entre eles, e assim por diante, até que cheguem no LSR não-adjacente de destino;
- c) cada LSR intermediário necessita abrir e processar as mensagens LDP trocadas entre os dois LSRs não-adjacentes das extremidades, pois deve alterar alguns TLVs destas mensagens de modo a encaminhá-las ao próximo LSR adjacente do caminho até que cheguem ao LSR não-adjacente de destino.

Por causa do processamento realizado pelos LSRs intermediários, soluções como o IPSec, SSL ou TLS não se aplicam ao caso. Para tratar esta situação, a solução proposta define mecanismos próprios ao protocolo LDP que possibilitam transportar os campos de autenticação transparentemente fim a fim através dos LSRs intermediários e dessa forma permitir que as extremidades do LSP, possam se autenticar mutuamente fim a fim [Müller].

A solução proposta faz uso de um mecanismo de assinatura digital da origem dos dados, baseado em criptografia e chaves assimétricas (chave pública e privada), anexado a cada mensagem LDP. Este mecanismo possibilita ao LSR receptor verificar e autenticar o originador das mensagens LDP.

A solução provê integridade de dados, através de um mecanismo de resumo de mensagens (hash) e adicionalmente também protege contra ataques de repetição através da inserção de nonce em cada mensagem LDP.

A solução não prove confidencialidade aos dados com base no fato de que informações transportadas nas mensagens LDP, o que pode ser comparado às informações transportadas por protocolos de distribuição de rotas IP como BGP, OSPF e outros, não são de natureza confidencial.

Como requisito, a solução proposta apenas exige que o LDP esteja operando no Modo de Controle Ordenado. Quanto aos modos de distribuição do LDP "Sob Demanda" e "Não Solicitado", ambos são compatíveis com a solução proposta. A solução foi projetada para ser utilizada com IP versão 4 (ipv4) e pode ser aplicada ao protocolo CR-LDP sem necessidade de adaptações [Müller].

2.3.2 - Modelo de Autenticação Proposto

Foram definidos dois novos Tlvs ao LDP para prover a autenticação, "Tlv de Nonce" e "Tlv de Hash", respectivamente. As mensagens LDP envolvidas no processo de autenticação são: Label Mapping, Label Request e LDP Notification.

Como um cenário de exemplo, mostrado na figura 21, temos um LERA que deseja estabelecer um LSP com um LERB. Considera-se que o LDP está configurado para operar no modo de distribuição Sob Demanda e Modo de Controle Ordenado.

O esquema abaixo ilustra o cenário, onde o LERB autentica positivamente a mensagem de requisição LDP enviada pelo LERA e retorna uma mensagem de mapeamento LDP autenticada ao LERA.

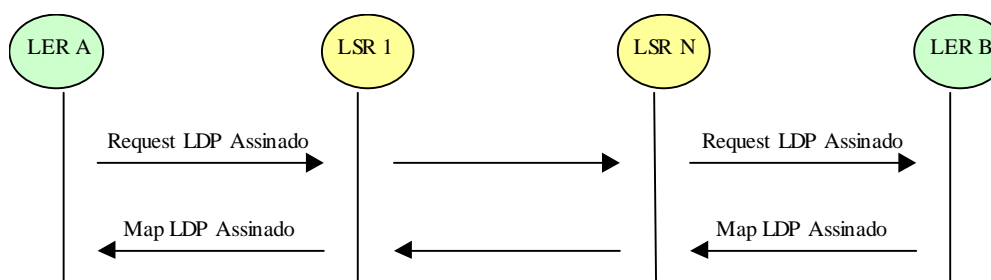


Figura 21 - Diagrama da autenticação fim a fim proposta para o LDP.

Ambos os LERs A e B fazem uso de um backbone MPLS. Tanto o LERA como o LERB confiam no backbone MPLS do provedor, porém desejam uma autenticação mútua entre eles para estabelecer um LSP. O LERA usa o Modo de Controle Ordenado do LDP e executa os seguintes passos:

- codifica uma mensagem de requisição LDP LABEL REQUEST;
- gera um nonce (por exemplo um *timestamp*);
- baseado no conteúdo desta mensagem gera um resumo aplicando um algoritmo de resumo de mensagens (*Hash*);
- o valor *Hash* é criptografado com a chave privada do LERA e incluído na mensagem LDP, juntamente com o seu identificador de LSR (LSR-ID);
- o LERA envia a mensagem LDP ao próximo LSR no caminho até o LERB.

Os LSRs do caminho (intermediários) que não podem atender a requisição solicitada repassam os campos da autenticação transparentemente até o LSR de egresso (LERB).

Quando o LERB recebe a mensagem de requisição LDP:

- verifica que o LERA é o originador da mensagem;
- verifica em sua configuração local se o LERA está autorizado a solicitar LSPs. Em caso positivo, seleciona a chave pública do LERA;
- decifra o valor Hash recebido na mensagem de requisição utilizando a chave pública do LERA, assim verifica a legitimidade da assinatura do LERA e pode ter certeza que a mensagem foi gerada por este LSR;

- da mesma forma que o LERA, gera um resumo da mensagem recebida (hash) e compara com o valor hash recebido, assim pode verificar a integridade da mensagem recebida;
- gera um nonce local (por exemplo um timestamp) e compara com o nonce recebido. Exemplos de formas de implementação de nonce podem ser: verificar se o tempo decorrido entre o envio e o recebimento do pacote ultrapassou a um valor estipulado (por exemplo 5 minutos); ou, ao enviar uma mensagem é gerado um número aleatório de natureza incremental e a cada troca de mensagem entre os LSRs envolvidos, este valor é incrementado segundo uma regra por eles conhecida e verificado na extremidade oposta, nesta forma cada extremidade deve armazenar o último valor nonce utilizado pelo parceiro para poder verificar se o incremento foi realizado da maneira correta [ABADI, 1996].

Se a autenticação ocorrer com sucesso, o LERB gera uma mensagem LDP LABEL MAPPING, incluindo a sua assinatura nos mesmos termos do LERA, e atribui um rótulo MPLS a FEC, conforme solicitado pelo LERA na mensagem de requisição.

Se a autenticação falhar, uma mensagem de notificação (LDP NOTIFICATION) deve ser enviada em resposta para reportar esta falha de autenticação (esta notificação opcionalmente também pode ser assinada pelo LERB).

Na mensagem de resposta, o LERB executa os mesmos passos que o LERA (gera um hash da mensagem e assina com a sua chave privada) e envia a resposta. Nos LSRs intermediários a resposta é repassada até alcançar o LERA. O LERA por sua vez detecta que é o Ingresso para FEC e procede a autenticação do LERB. Para isso, verifica se o LERB está autorizado a estabelecer o LSP, verifica a assinatura e a integridade da mensagem recebida e executa ou não o estabelecimento do LSP com o LERB, baseado no resultado da autenticação.

2.3.3 - Novos TLVs Definidos

Foi necessária a criação novos TLVs no LDP para concretizar a solução de autenticação: TLV de Hash e TLV de Nonce.

2.3.3.1 - TLV de Hash

Foi definido um novo TLV para transportar um valor hash encriptado.

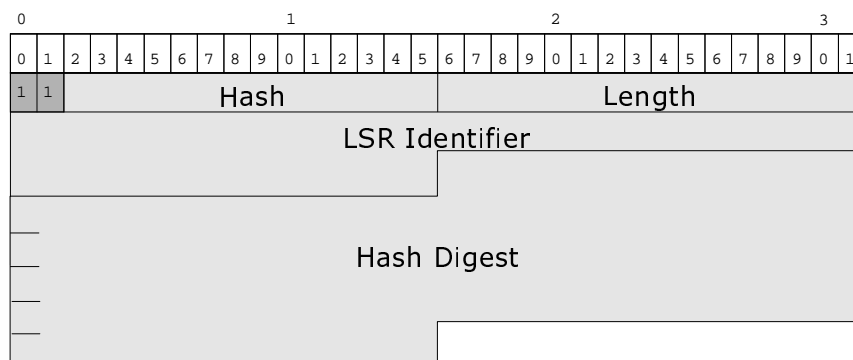


Figura 22 - TLV de Hash

U-bit: (1 bit) O bit “*unknow-bit*” deve ser atribuído em 1 porque os LSRs intermediários que não conhecem este novo TLV devem ignorá-lo e processar o resto da mensagem.

F-bit: (1 bit) O bit “*forward-bit*” deve ser atribuído em 1 porque os LSRs intermediários que não conhecem este novo TLV devem repassá-lo de forma transparente para o próximo LSR do caminho.

Hash: (14 bits) Tipo do TLV. Precisa ser definido um código de tipo para este novo TLV dentro do LDP, conforme a RFC 3036 [ANDERSON, 2001].

Length: (2 bytes) Este campo indica o tamanho total em bytes dos seguintes campos:

LSR Identifier: (6 bytes) Este campo contém o identificador da entidade LSR que originou a mensagem LDP. O campo é composto pelo LSR-ID e pelo espaço de rótulos (*labels*) usado pelo LSR. O receptor da mensagem utiliza este campo no processo de identificação da origem.

Hash Digest: (20 bytes) Este campo contém o valor HASH gerado a partir de uma mensagem LDP, conforme descrito na sessão **Erro! A origem da referência não foi encontrada.** O TLV de HASH deve obrigatoriamente ser anexado ao final da mensagem LDP, como o último TLV da mensagem. Este valor *hash* será encriptado usando a chave privada do remetente da mensagem LDP. Para definir o tamanho deste campo, foram considerados os

algoritmos de *hash* (sha-1, 160 bits) e de criptografia assimétrica "Curvas Elípticas", conforme discutido na seção 0. Neste caso a função hash gera uma saída de 20 bytes e a função de criptografia assimétrica aplicada sobre este valor hash também gera uma saída de 20 bytes, sendo este o tamanho necessário para o campo "Hash Digest".

2.3.3.2 - TLV de Nonce

Foi definido um novo TLV para transportar o valor do *nonce*.

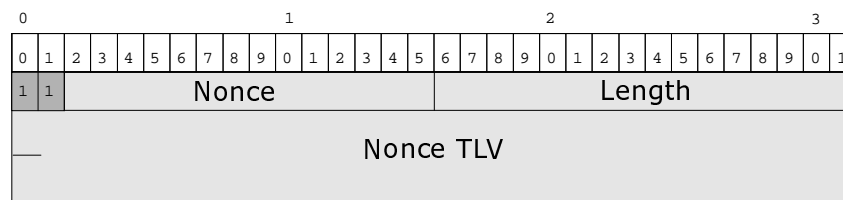


Figura 23 – TLV de Nonce

U-bit e F-bit seguem a mesma descrição anterior (item **Erro! A origem da referência não foi encontrada.**).

Nonce: (14 bits) Tipo do TLV. Precisa ser definido um código de tipo para este novo TLV, conforme a RFC 3036 [ANDERSON, 2001].

Length: (2 bytes) Indica o tamanho em bytes do campo "Nonce Value".

Nonce Value: (8 bytes) Este campo contém um valor único que deve ser incrementado a cada mensagem trocada entre dois LSRs. Um valor do tipo *timestamp* é um exemplo de contador de natureza incremental. O TLV de Nonce pode ser inserido em qualquer posição dentro da mensagem LDP, porém antes do TLV de Hash.

2.3.3.3 - Novo Código de Status "Authentication Failed"

A proposta define um novo Código de Status (*Status Code*) com o valor "Authentication Failed" para o TLV de Status do LDP. Este Código de Status será usado nas mensagens LDP NOTIFICATION, para anunciar que uma mensagem LABEL MAPPING ou LABEL REQUEST recebida falhou na autenticação.

Os campos “U-bit” e “F-bit” dentro do TLV de Status do LDP (*Status TLV*) devem ser atribuídos em “1” de forma a habilitar o encaminhamento transparente do TLV de Status através dos LSRs intermediários.

No campo “Status Code” no TLV de Status do LDP o *flag* “E-bit” pode ser atribuído em “0” ou “1” dependendo da política local do LSR. O *flag* “F-bit” deve ser atribuído em “1” para ficar alinhado com o campo “F-bit” do TLV de Status do LDP.

2.3.3.4 - Considerações sobre o Registro dos TLVs

Tendo em vista que o LDP é um protocolo regulamentado pelo IETF (*Internet Engineering Task Force*) todas as definições e tipos estão registrados no IANA (*Internet Assigned Number Authority*). Para ficar conforme com a definição do LDP (RFC 3036) valores de tipos devem ser registrados para os novos TLVs definidos na proposta, respectivamente, TLV de Hash e TLV de Nonce.

Será necessário também registrar um novo “Código de Status” (*Status Code*) com o valor “Authentication Failed” para o TLV de Status do LDP, conforme RFC 3036, o qual será usado nas mensagens LDP NOTIFICATION para anunciar que uma mensagem LABEL MAPPING ou LABEL REQUEST recebida falhou na autenticação.

2.3.4 - Procedimentos da Autenticação Fim a Fim

Os TLVs da autenticação devem ser inseridos nas mensagens LDP LABEL MAPPING ou LABEL REQUEST, somente se o LSR for o EGRESSO ou INGRESSO para uma das FEC em questão na mensagem LDP.

Opcionalmente podem ser inseridos nas mensagens LDP NOTIFICATION, quando for notificada a falha de autenticação de uma mensagem LDP LABEL REQUEST ou LABEL MAPPING.

2.3.4.1 - Procedimentos de Envio de Mensagens LDP

Tanto no modo de distribuição SOB DEMANDA como no modo de distribuição NÃO SOLICITADO os TLVs da autenticação devem ser anexados às mensagens LDP nos seguintes casos:

- em mensagens LABEL REQUEST, quando o emissor detectar que é o INGRESSO para alguma das FECs da mensagem;
- em mensagens LABEL MAPPING, quando o emissor detectar que é o EGRESSO para alguma das FECs da mensagem;
- em mensagens LDP NOTIFICATION para notificar a falha de autenticação de uma mensagem LABEL REQUEST ou LABEL MAPPING recebida.

Para codificar os TLVs da autenticação, considerando os relógios sincronizados, o emissor gera um valor nonce (por exemplo um *timestamp* baseado na hora local) e codifica o TLV de Nonce anexando-o ao final da mensagem LDP. Depois disso codifica o TLV de HASH seguindo os seguintes passos:

- no campo "LSR Identifier" o emissor anexa o seu LSR-ID e o espaço de rótulos (*labels*) em uso;
- a composição do campo "Hash Digest" depende do tipo da mensagem LDP.

Para mensagens LABEL REQUEST e LABEL MAPPING a entrada de dados é formada pela cadeia de bytes conforme descrito na figura seguinte:

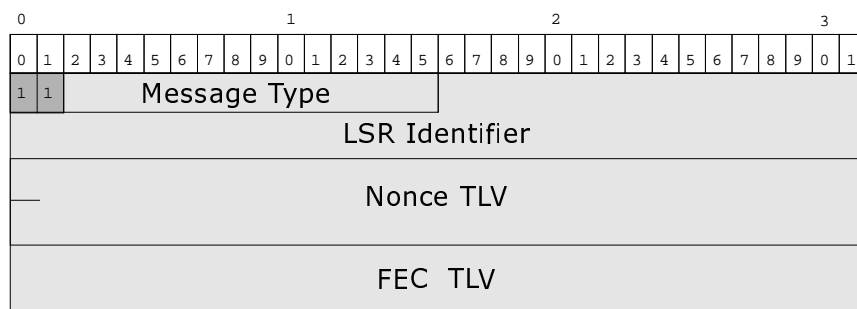


Figura 23 - Entradas para mensagens LDP LABEL REQUEST e LABEL MAPPING

Para mensagens LDP NOTIFICATION a entrada de dados é formada pela cadeia de bytes conforme descrito a seguir:

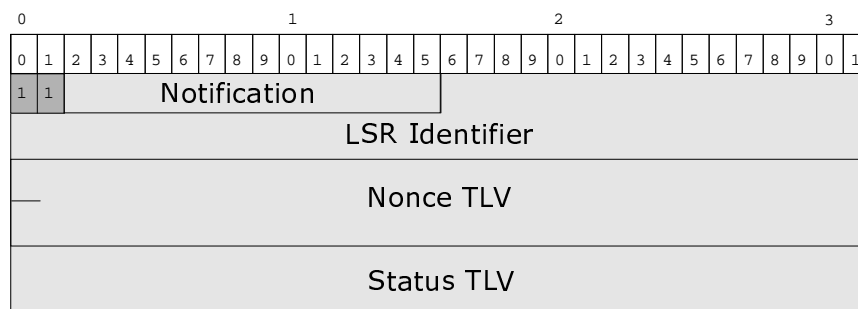


Figura 24 - Entrada para mensagens LDP NOTIFICATION

Por final a mensagem LDP é enviada ao próximo LSR do caminho (*next hop*).

2.3.4.2 - Procedimentos ao RECEBER Mensagens LDP

Ao receber uma mensagem LDP LABEL MAPPING ou LABEL REQUEST, tanto no modo de distribuição SOB DEMANDA como no modo de distribuição NÃO SOLICITADO, o receptor deve processar os TLVs da autenticação apenas nos seguintes casos:

- em mensagens LABEL REQUEST, quando o receptor detectar que é o EGRESSO para alguma da(s) FEC(s) da mensagem;
- em mensagens LABEL MAPPING, quando o receptor detectar que é o INGRESSO para alguma da(s) FEC(s) da mensagem;
- em mensagens LDP Notification quando o receptor detectar que é o INGRESSO ou EGRESSO para alguma da(s) FEC(s) da mensagem.

Se o LSR for intermediário para a FEC em questão, o mesmo deve repassar os campos da autenticação transparentemente ao próximo LSR do caminho e assim por diante até o LSR de Ingresso ou Egresso do LSP onde os Tlvs da autenticação serão processados.

Para verificar os TLVs da autenticação o LSR deve proceder da seguinte maneira:

- a) identificar o emissor da mensagem através do campo "LSR Identifier" do TLV de Hash;
- b) verificar em sua configuração local se este LSR é autorizado, e em caso positivo selecionar a sua chave pública;
- c) decifrar o campo "Hash Value" do TLV de Hash recebido usando a chave pública do remetente, de modo a validar a assinatura digital. Se a operação for realizada com sucesso significa que o remetente é autêntico;
- d) gerar um valor Hash da mensagem recebida. Comparar este *hash* com o valor do campo "Hash Digest" do TLV de Hash recebido, que foi decifrado com a chave pública do remetente. Se os valores forem iguais significa que a mensagem está íntegra;
- e) considerando os relógios sincronizados, deve gerar um *timestamp* baseado na sua hora local e comparar com o *timestamp* recebido (campo "Nonce Value" do TLV de Nonce) de modo a verificar a quantidade de tempo que se passou entre o envio e o recebimento da mensagem. Se este intervalo de tempo for superior a um tempo estipulado (por exemplo 10 minutos) a mensagem deve ser descartada.

Se a verificação da autenticação falhar em alguma destas etapas, deve ser retornada uma mensagem LDP NOTIFICATION com o código de status "Authentication Failed". Opcionalmente esta notificação pode incluir os TLVs da autenticação de modo que o LSR receptor (que enviou a última mensagem LDP LABEL MAPPING ou LABEL REQUEST que falhou) possa autenticar a origem e verificar a integridade da mensagem de notificação retornada.

Nos demais casos em que o LSR não é o EGRESSO nem o INGRESSO para a(s) FEC(s) da mensagem LDP recebida, este deve repassar os TLVs da autenticação para o próximo LSR do caminho mantendo a ordem destes TLVs dentro da nova mensagem LDP resultante, conforme a mensagem original recebida.

2.3.5 - Discussão sobre a Solução de Autenticação Proposta

A solução proposta provê autenticação da origem, integridade e controle contra ataques de repetição as mensagens LDP trocadas entre duas entidades LSR durante o estabelecimento de um LSP.

O escopo de aplicação da solução está voltado a fornecer uma solução de autenticação fim a fim para viabilizar que o LSR de Ingresso e o respectivo LSR de Egresso de um LSP possam se autenticar mutuamente durante o estabelecimento do primeiro LSP entre ambos.

O tipos de mensagens LDP utilizados para prover o mecanismo de autenticação fim a fim ao LDP são: LABEL REQUEST, LABEL MAPPING e LDP NOTIFICATION. Estes três tipos de mensagens fornecem condições ao LDP solicitar e atribuir rótulos para o estabelecimento de LSPs, e notificar falhas referentes a estas operações.

A solução requer que o LDP opere no Modo de Controle ORDENADO e que a ordenação dos TLVs dentro das mensagem LDP não seja alterada pelas entidades LSR intermediárias que vão repassar os TLVs da autenticação. Quanto ao Modo de Distribuição do LDP, ambos os modos SOB DEMANDA e NÃO SOLICITADO são compatíveis com a solução proposta.

O campo “LSR Identifier” no TLV de Hash tem por objetivo identificar para o receptor da mensagem, a entidade LSR que efetivamente originou a mensagem.

2.3.5.1 - Distribuição de Chaves

O método de autenticação proposto é baseado em assinatura digital, sendo que cada LSR envolvido precisa gerar/conhecer seu próprio par de chaves (pública e privada) e ambas as entidades LSR das extremidades do LSP (Ingresso e Egresso) devem conhecer a chave pública do LSR da extremidade oposta de modo a poder validar sua assinatura digital.

Sugere-se duas alternativas para distribuir estas chaves públicas:

- informar manualmente na configuração local de cada LSR as chaves públicas dos LSRs autorizados. Na parte de implementação deste trabalho foi adotada esta solução de distribuição de chaves;
- utilizar certificação digital.

2.3.5.2 - Integridade da Mensagem

O emissor codifica uma mensagem LDP e gera um resumo da mesma (Hash). Este valor hash é inserido no campo "HASH Value" do TLV de Hash. O receptor executa o mesmo procedimento sobre a mensagem recebida e compara com o valor hash recebido do emissor. O valor hash não pode ser alterado durante a comunicação, pois o mesmo é cifrado com a chave privada do emissor.

2.3.5.3 - Autenticação da Origem

O emissor cifra o campo "Hash Digest" do TLV de Hash usando a sua chave privada e anexa seu LSR-ID no campo "LSR Identifier". Com base no campo "LSR Identifier" o receptor da mensagem pode selecionar a chave pública do emissor e decifrar o valor hash recebido. Se conseguir decifrar a mensagem com sucesso, comprova que o emissor é autêntico.

2.3.5.4 - Controle contra Ataques de Repetição

O mecanismo de controle contra ataques por repetição é provido através de um valor nonce. Este nonce é adicionado a mensagem LDP pelo emissor, dentro do campo "Nonce Value" do TLV de Nonce.

O valor do nonce deve ser de natureza incremental, como por exemplo, um timestamp. No caso de um timestamp deve existir um método de sincronismo de relógio entre as entidades que estão se comunicando. Neste caso o emissor gera um timestamp baseado no relógio local e envia o nonce, o receptor executa o mesmo procedimento localmente e depois compara os dois valores timestamp, se o tempo decorrido entre o envio e o recebimento da mensagem for superior a um tempo estipulado (por exemplo 10 minutos) a mensagem deve ser descartada.

Uma vez que este TLV de Nonce está incluso no Hash da mensagem, ele não pode ser adulterado durante a comunicação. Por ser de natureza incremental impede a retransmissão desta ou de outra mensagem com este mesmo TLV de Nonce.

2.3.6 - Considerações sobre Algoritmos de Criptografia

Para função Hash, o algoritmo sugerido na solução é o SHA-1 (Secure Hash Algorithm), com digest de 160 bits, ou seja, o algoritmo gera como saída uma string com 20 bytes de tamanho.

Para as funções de criptografia assimétrica, os algoritmos sugeridos são Curvas Elípticas e RSA (Rivest-Shamir-Adleman). Literatura sobre ambos pode ser encontrada em [STALLINGS, 1999].

Com o objetivo de gerar o mínimo *overhead* possível ao protocolo LDP, sugere-se o uso do algoritmo de Curvas Elípticas, o qual tem como vantagens ser rápido e trabalhar com blocos pequenos, sendo o mais adequado para a proposta de autenticação.

O RSA está sendo sugerido como uma opção alternativa para a solução proposta, pois é um algoritmo bastante difundido e amplamente utilizado nas atuais soluções de criptografia digital.

A configuração padrão do RSA sugere um tamanho de chave de 1024 bits, considerando que a entrada será uma string com 20 bytes de tamanho, o resultado da função RSA será uma string criptografada com 128 bytes de tamanho.

Como alternativa para diminuir o tamanho da saída gerada pela função RSA, sugere-se utilizar uma chave de 512 bits de tamanho que resulta em uma string criptografada de 64 bytes de tamanho. Este tamanho de chave, 512 bits, fornece um nível de segurança aceitável para a solução de autenticação proposta.

Na solução proposta o Tlv de Hash foi definido para armazenar 20 bytes no campo "Hash Digest". Se o RSA com chave de 512 bits, for adotado como solução, este campo deve ser expandido para armazenar 64 bytes.

Considerando a regra de nomenclatura de algoritmos de assinatura digital, geralmente composta pelo nome do algoritmo de função *hash* seguido do nome do algoritmo de criptografia assimétrica, a solução sugerida como a mais indicada nesta proposta é a assinatura digital "Sha1 Curvas Elípticas", composta pela algoritmo "SHA1, 160 bits" para a função *hash* e o algoritmo de "Curvas Elípticas" para a função de criptografia assimétrica [Müller].

2.3.7 - Distribuição de Chaves usando Certificação Digital

O emprego de chaves locais, informadas manualmente na configuração de cada LSR, levanta uma problemática comum nos ambientes distribuídos que é a distribuição de chaves. Nesse contexto o gerenciamento da solução é um dos aspectos mais comprometidos.

Uma das formas de contornar este problema é a utilização de certificação digital, que além de fornecer uma solução automatizada para distribuição das chaves públicas no ambiente da solução, fornece um nível de segurança superior.

Tendo em vista que roteadores (LSRs) têm pouca memória de armazenamento, geralmente reduzido a uma memória "Flash ROM", a solução deve contemplar esta restrição. Um certificado digital ocupa em média 1 Kbyte de armazenamento, dessa forma quanto a este aspecto a certificação digital é viável.

A solução proposta, definida em [Müller], prevê a especificação da utilização de autoridades certificadoras como forma de distribuição de chaves públicas entre os LSRs. A utilização de autoridades certificadoras é definido como trabalho futuro e portanto não implementada no protótipo.

2.4 - CONTRIBUIÇÕES EFETIVAS

A contribuição efetiva se deu na implementação do protótipo que serviu para validar a solução de autenticação. Para a implementação do protótipo, utilizou-se a linguagem de programação C e as ferramentas, tecnologias e algoritmos demonstrados no capítulo 2.1.

Alguns algoritmos e definições utilizados no protótipo não são os mesmos que os especificados no projeto final, pois o protótipo foi sendo desenvolvido ao mesmo tempo em que a solução estava sendo aprimorada.

A execução do protótipo considera que os relógios dos LSRs do ambiente estão sincronizados. Dessa forma no ambiente de testes os relógios foram sincronizados manualmente.

Como valor *nonce* foi adotado um *timestamp*. O tempo de vida deste *nonce* foi estipulado em 10 minutos. Dessa forma o receptor da mensagem gera um valor *timestamp* baseado na hora local e compara com o valor *nonce* recebido, de forma a verificar se o prazo de expiração (10 minutos) entre o envio e o recebimento da mensagem não esgotou.

O algoritmo de hash utilizado foi o SHA-1 (160 bits), o qual gera como saída um *digest* de 160 bits, ou 20 bytes.

O algoritmo de Curvas Elípticas foi definido como a melhor opção de algoritmo assimétrico para o ambiente, porém na implementação foi utilizado o RSA com chave de 1024 bits, o qual gera uma saída de 128 bytes, considerando que a entrada são 20 bytes, resultado da função *hash* executada. Dessa forma o tamanho do campo "Hash Digest" do Tlv de Hash, foi estendido para 128 bytes.

Para implementar o controle de acesso dos pares que tem permissão de trocar mensagens LDP com um LSR que implementa a autenticação fim a fim do LDP, foi adotado um arquivo que inclui os pares LDP que precisam se autenticar e suas respectivas chaves públicas, localizado em `/usr/local/etc/ldp_peers`. Esta política foi adotada de modo a facilitar a adaptação da implementação proposta ao ambiente do protótipo. Dessa forma se o LSR originador da mensagem LABEL MAPPING ou LABEL REQUEST estiver incluso neste arquivo de controle de acesso, a autenticação será validada para este LSR remoto. Se o mesmo não estiver incluso nesta lista, a autenticação não será validada e os Tlvs da autenticação serão ignorados durante o processamento da mensagem LDP recebida.

Cada LSR possui também sua chave pública e privada nos respectivos arquivos (`/usr/local/etc/ldp_key` e `/usr/local/etc/ldp_key.pub`). A geração desse par de chaves foi realizada manualmente em cada LSR do ambiente de testes. As chaves são compatíveis com o algoritmo RSA e possuem 1024 bits de tamanho.

2.4.2 - Implementação da Solução de Autenticação

O objetivo desta seção é dar uma visão de como foi realizada a implementação do protótipo deste trabalho. A autenticação fim a fim proposta para o LDP foi inserida ao pacote LDP-PORTABLE através da alteração de seu código fonte.

O código fonte do LDP-PORTABLE está escrito em "C ANSI" e depois de compilado, gera um arquivo executável "**mplsd**" localizado por *default* no diretório `/usr/local/sbin` do linux. Os fontes do LDP-PORTABLE são constituídos basicamente por dois componentes principais:

- **ldp-portable/lib**: são as bibliotecas usadas pelo LDP;

- **ldp-portable/zebra-ldp.diff** : o *patch* que porta o LDP para ZEBRA.

O código fonte completo da autenticação fim a fim implementada, com todas as alterações efetuadas, ficará disponível publicamente para consultas. Todos os blocos de código alterados por esta implementação estão assinalados no código fonte com blocos de marcação de início e fim, conforme abaixo:

```
// BEGIN: AUTH
...
código inserido ou alterado pela autenticação fim a fim.
...
// END: AUTH
```

Nesta seção serão apresentadas apenas as partes principais da implementação realizada. Para ter acesso a todas as alterações efetuadas por esta implementação será necessário examinar o código fonte.

A implementação da autenticação proposta neste trabalho basicamente gerou alterações e inserções de código nas bibliotecas do LDP-PORTABLE. Foram alterados os seguintes arquivos de biblioteca do código fonte do LDP-PORTABLE, localizados no diretório "ldp-portable/lib": ldp_nortel.h, ldp_nortel.c, ldp_struct.h, ldp_pdu_setup.h, ldp_pdu_setup.c, ldp_label_mapping.c, ldp_label_request.c, ldp_state_machine.c, ldp_notif.c. Além destas alterações, foram criados dois novos arquivos de biblioteca que são: ldp_auth.c e ldp_auth.h.

- os arquivos ldp_nortel.c e ldp_nortel.h contém a implementação que faz a codificação e decodificação de todos os tipos de mensagens do LDP e de todos os tipos de TLVs usados no LDP;
- o arquivo ldp_struct.h possui as estruturas usadas pelo ldp-portable, exceto estruturas de mensagens e estruturas de TLV;
- o arquivo ldp_label_mapping.c possui o código responsável por mensagens de atribuição de labels (label mapping);
- o arquivo ldp_notif.c possui o código responsável por mensagens de notificação (LDP notification);
- o arquivo ldp_label_request.c possui o código responsável por mensagens de requisição de labels ("label request");
- o arquivo ldp_pdu_setup.c é responsável pelo preenchimento dos campos dos diversos TLV's do LDP;

- O arquivo `ldp_state_machine.c` contém a implementação dos eventos do LDP.

A seguir serão listadas as principais alterações efetuadas pela implementação da autenticação fim a fim dentro do código fonte do LDP-PORTABLE juntamente com uma descrição do significado das funções e/ou blocos de código inseridos.

a) alterações efetuadas no arquivo "ldp-portable/lib/ldp_nortel.h":

Foram definidos os valores de tipo para os TLVs de Hash e Nonce. Estes valores são utilizado no campo "Type" dos TLVs e tem função de identificar o tipo do TLV dentro da mensagem LDP.

```
#define MPLS_HASH_TLVTYPE      0x0880
#define MPLS_NONCE_TLVTYPE    0x0881
```

Foi definido o tamanho fixo de cada TLV, sem o tamanho do cabeçalho do TLV.

```
#define MPLS_NONCETLV_FIXLEN   4
#define MPLS_HASHTLV_FIXLEN   134
```

Foram definidos valores de retorno para as funções de codificação e decodificação do TLV de Nonce e TLV de Hash.

```
#define MPLS_ENC_NONCEERROR    -81
#define MPLS_DEC_NONCEERROR    -82
#define MPLS_ENC_HASHERROR     -83
#define MPLS_DEC_HASHERROR     -84
```

Foi definida a estrutura do TLV de Nonce. A estrutura "baseTlv" armazena o cabeçalho LDP, comum a todos TLVs LDP e o campo "nonce" armazena o valor do *nonce*, ou seja, o *timestamp* gerado pelo LSR ao enviar um TLV de Nonce.

```
typedef struct mplsLdpNonceTlv_s {
    struct mplsLdpTlv_s baseTlv;
    u_int nonce;
} mplsLdpNonceTlv_t;
```

Foi definida a estrutura do TLV de Hash. A estrutura "baseTlv" armazena o cabeçalho LDP, comum a todos TLVs LDP. O campo "lSrAddress" armazena o Identificador do LSR (LSR-ID) e o espaço de labels usado por este LSR . O campo "hashDigest" armazena o resultado da função *hash* sha-1/160 bits aplicada a mensagem corrente. Este valor será encriptado com a chave privada do LSR no envio e decriptada com a chave pública do LSR origem na recepção do TLV.

```
typedef struct mplsLdpHashTlv_s {
    struct mplsLdpTlv_s baseTlv;
```

```

    u_int lsrAddress;
    u_short labelSpace;
    u_char hashDigest[128];
} mplslDpHashTlv_t;

```

Foram alteradas as estruturas da mensagem Label Mapping e Label Request. Na estrutura da mensagem Label Mapping foi inserida a estrutura "nonceTlv", que armazena os campos do TLV de Nonce, e estrutura "hashTlv", que armazena os campos do TLV de Hash. Os campos *booleanos* "nonceTlvExists" e "hashTlvExists" servem para indicar se os TLVs de Nonce e Hash estão presentes na mensagem LDP Label Mapping.

```

typedef struct mplslDpLblMapMsg_s {
    ...
    // BEGIN: AUTH
    struct mplslDpNonceTlv_s nonceTlv;
    struct mplslDpHashTlv_s hashTlv;
    // END: AUTH
    ...
    // BEGIN: AUTH
    u_char nonceTlvExists:1;
    u_char hashTlvExists:1;
    // END: AUTH
} mplslDpLblMapMsg_t;

```

Na estrutura da mensagem Label Request foi inserida a estrutura "nonceTlv", que armazena os campos do TLV de Nonce, e a estrutura "hashTlv", que armazena os campos do TLV de Hash. Os campos *booleanos* "nonceTlvExists" e "hashTlvExists" servem para indicar se os TLVs de Nonce e Hash estão presentes na mensagem Label Request.

```

typedef struct mplslDpLblReqMsg_s {
    ...
    // BEGIN: AUTH
    struct mplslDpNonceTlv_s nonceTlv;
    struct mplslDpHashTlv_s hashTlv;
    // END: AUTH
    ...
    // BEGIN: AUTH
    u_char nonceTlvExists:1;
    u_char hashTlvExists:1;
    // END: AUTH
} mplslDpLblReqMsg_t;

```

O arquivo ldp_nortel.h também define os protótipos das funções utilizada no arquivo ldp-portable/lib/ldp_nortel.c.

b) alterações efetuadas no arquivo "ldp-portable/lib/ldp_nortel.c":

O arquivo ldp_nortel.c executa os procedimentos de codificação e decodificação dos TLVs e mensagens LDP e os respectivos procedimentos de impressão na saída padrão com a finalidade de fornecer um log da execução.

O procedimento de codificação da mensagem Label Mapping foi modificado, inserindo os procedimentos necessários para codificar e inserir os TLVs de Nonce e Hash, caso sejam solicitados (hashTlvExists e nonceTlvExists atribuídos em 1).

```
int Mpls_encodeLdpLblMapMsg
(mplsLdpLblMapMsg_t * lblMapMsg, u_char * buff, int bufSize) {
    ...
    // BEGIN : AUTH
    if (lblMapMsgCopy.nonceTlvExists) {
        encodedSize = Mpls_encodeLdpNonceTlv(&(lblMapMsgCopy.nonceTlv), \
            tempBuf, bufSize - totalSize);
        if (encodedSize < 0) {
            return MPLS_ENC_NONCEERROR;
        }
        PRINT_OUT("Encoded for Nonce Tlv %d bytes\n", encodedSize);
        tempBuf += encodedSize;
        totalSize += encodedSize;
    }
    if (lblMapMsgCopy.hashTlvExists) {
        encodedSize = Mpls_encodeLdpHashTlv(&(lblMapMsgCopy.hashTlv),
            tempBuf, bufSize - totalSize);
        if (encodedSize < 0) {
            return MPLS_ENC_HASHERROR;
        }
        PRINT_OUT("Encoded for Hash Tlv %d bytes\n", encodedSize);
        tempBuf += encodedSize;
        totalSize += encodedSize;
    }
    // END: AUTH
    ...
} /* End: Mpls_encodeLdpLblMapMsg */
```

O procedimento de decodificação da mensagem Label Mapping também foi modificado. Foi inserido o código de decodificação dos TLVs de Nonce e Hash caso eles existam na mensagem (hashTlvExists e nonceTlvExists atribuídos em 1):

```
int Mpls_decodeLdpLblMapMsg
(mplsLdpLblMapMsg_t * lblMapMsg, u_char * buff, int bufSize) {
    ...
    // BEGIN : AUTH
    case MPLS_NONCE_TLVTYPE:
    {
        decodedSize = Mpls_decodeLdpNonceTlv(&(lblMapMsg->nonceTlv),
            tempBuf, bufSize - totalSize);
        if (decodedSize < 0) {
            PRINT_ERR("Failure when decoding nonce tlv from LblReq
msg\n");
            return MPLS_DEC_NONCEERROR;
        }
        PRINT_OUT("Decoded for nonce tlv %d bytes\n", decodedSize);
        tempBuf += decodedSize;
        totalSize += decodedSize;
        totalSizeParam += decodedSize;

        lblMapMsg->nonceTlvExists = 1;
        lblMapMsg->nonceTlv.baseTlv = tlvTemp;
        break;
    }
}
```

```

case MPLS_HASH_TLVTYPE:
{
    decodedSize = Mpls_decodeLdpHashTlv(&(lblMapMsg->hashTlv),
        tempBuf, bufSize - totalSize);
    if (decodedSize < 0) {
        PRINT_ERR("Failure when decoding hash tlv from LblReq msg\n");
        return MPLS_DEC_HASHERROR;
    }
    PRINT_OUT("Decoded for hash tlv %d bytes\n", decodedSize);
    tempBuf += decodedSize;
    totalSize += decodedSize;
    totalSizeParam += decodedSize;

    lblMapMsg->hashTlvExists = 1;
    lblMapMsg->hashTlv.baseTlv = tlvTemp;
    break;
}
// END : AUTH
...
} /* End: Mpls_decodeLdpLblMapMsg */

```

O procedimento de codificação da mensagem Label Request foi modificado inserindo os procedimentos necessários para inserir os TLVs de Nonce e Hash, caso sejam solicitados (hashTlvExists e nonceTlvExists atribuídos em 1).

```

int Mpls_encodeLdpLblReqMsg
(mplsLdpLblReqMsg_t * lblReqMsg, u_char * buff, int bufSize) {
...
// BEGIN: AUTH
if (lblReqMsgCopy.nonceTlvExists) {
    encodedSize = Mpls_encodeLdpNonceTlv(
&(lblReqMsgCopy.nonceTlv), tempBuf, bufSize - totalSize);
    if (encodedSize < 0) {
        return MPLS_ENC_NONCEERROR;
    }
    PRINT_OUT("Encoded for Nonce Tlv %d bytes\n", encodedSize);
    tempBuf += encodedSize;
    totalSize += encodedSize;
}
if (lblReqMsgCopy.hashTlvExists) {
    encodedSize = Mpls_encodeLdpHashTlv(&(lblReqMsgCopy.hashTlv),
        tempBuf, bufSize - totalSize);
    if (encodedSize < 0) {
        return MPLS_ENC_HASHERROR;
    }
    PRINT_OUT("Encoded for Hash Tlv %d bytes\n", encodedSize);
    tempBuf += encodedSize;
    totalSize += encodedSize;
}
// END : AUTH
...
} /* End: Mpls_encodeLdpLblReqMsg */

```

O procedimento de decodificação da mensagem Label Request também foi modificado. Foi inserido o código de decodificação dos TLVs de Nonce e Hash caso eles existam (hashTlvExists e nonceTlvExists atribuídos em 1):

```

int Mpls_decodeLdpLblReqMsg(mplsLdpLblReqMsg_t * lblReqMsg, u_char * buff,
int bufSize) {
    ...
    //BEGIN: AUTH
    case MPLS_NONCE_TLVTYPE:
    {
        decodedSize = Mpls_decodeLdpNonceTlv(&(lblReqMsg->nonceTlv),
        tempBuf, bufSize - totalSize);
        if (decodedSize < 0) {
            PRINT_ERR("Failure when decoding nonce tlv from LblReq
msg\n");
            return MPLS_DEC_NONCEERROR;
        }
        PRINT_OUT("Decoded for nonce tlv %d bytes\n", decodedSize);
        tempBuf += decodedSize;
        totalSize += decodedSize;
        totalSizeParam += decodedSize;

        lblReqMsg->nonceTlvExists = 1;
        lblReqMsg->nonceTlv.baseTlv = tlvTemp;
        break;
    }
    case MPLS_HASH_TLVTYPE:
    {
        decodedSize = Mpls_decodeLdpHashTlv(&(lblReqMsg->hashTlv),
        tempBuf, bufSize - totalSize);
        if (decodedSize < 0) {
            PRINT_ERR("Failure when decoding hash tlv from LblReq msg\n");
            return MPLS_DEC_HASHERROR;
        }
        PRINT_OUT("Decoded for hash tlv %d bytes\n", decodedSize);
        tempBuf += decodedSize;
        totalSize += decodedSize;
        totalSizeParam += decodedSize;

        lblReqMsg->hashTlvExists = 1;
        lblReqMsg->hashTlv.baseTlv = tlvTemp;
        break;
    }
}
//END: AUTH
...
} /*End: Mpls_decodeLdpLblReqMsg */

```

Foi criado um novo procedimento para a codificação do TLV de Nonce:

```

int Mpls_encodeLdpNonceTlv \
(mplsLdpNonceTlv_t * nonceTlv, u_char * buff, int bufSize) {
    int encodedSize = 0;
    u_char *tempBuf = buff; /* no change for the buff ptr */
    u_char *nonceTlvPtr;

    if (MPLS_TLVFIXLEN + MPLS_NONCETLV_FIXLEN > bufSize) {
        /* not enough room */
        return MPLS_ENC_BUFFTOOSMALL;
    }

    /*
    * encode for tlv
    */
    encodedSize = Mpls_encodeLdpTlv(&(nonceTlv->baseTlv),
        tempBuf, MPLS_TLVFIXLEN);
    if (encodedSize < 0) {
        return MPLS_ENC_TLVERROR;
    }
    tempBuf += encodedSize;

    nonceTlvPtr = (u_char *) nonceTlv;

```

```

nonceTlvPtr += encodedSize;

MEM_COPY(tempBuf, nonceTlvPtr, MPLS_NONCETLV_FIXLEN);

return (MPLS_TLVFIXLEN + MPLS_NONCETLV_FIXLEN);
}
/* End: Mpls_encodeLdpNonceTlv */

```

Foi criado um novo procedimento para decodificação do TLV de Nonce:

```

int Mpls_decodeLdpNonceTlv
(mplsLdpNonceTlv_t * nonceTlv, u_char * buff, int bufSize) {
    u_char *nonceTlvPtr;

    if (MPLS_NONCETLV_FIXLEN > bufSize) {
        PRINT_ERR("failed decoding nonce tlv\n");
        return MPLS_DEC_BUFFTOOSMALL;
    }
    nonceTlvPtr = (u_char *) nonceTlv;
    nonceTlvPtr += MPLS_TLVFIXLEN;

    MEM_COPY(nonceTlvPtr, buff, MPLS_NONCETLV_FIXLEN);

    return MPLS_NONCETLV_FIXLEN;
}
/* End: Mpls_decodeLdpNonceTlv */

```

Foi criado um novo procedimento para a codificação do TLV de Hash:

```

int Mpls_encodeLdpHashTlv
(mplsLdpHashTlv_t * hashTlv, u_char * buff, int bufSize) {
    int encodedSize = 0;
    u_char *tempBuf = buff; /* no change for the buff ptr */
    u_char *hashTlvPtr;

    if (MPLS_TLVFIXLEN + MPLS_HASHTLV_FIXLEN > bufSize) {
        /* not enough room */
        return MPLS_ENC_BUFFTOOSMALL;
    }

    /*
     * encode for tlv
     */
    encodedSize=Mpls_encodeLdpTlv(&(hashTlv->baseTlv),tempBuf,
MPLS_TLVFIXLEN);
    if (encodedSize < 0) {
        return MPLS_ENC_TLVERROR;
    }
    tempBuf += encodedSize;

    hashTlvPtr = (u_char *) hashTlv;
    hashTlvPtr += encodedSize;

    MEM_COPY(tempBuf, hashTlvPtr, MPLS_HASHTLV_FIXLEN);

    return (MPLS_TLVFIXLEN + MPLS_HASHTLV_FIXLEN);
}
/* End: Mpls_encodeLdpHashTlv */

```

Foi criado um novo procedimento para decodificação do TLV de Hash:

```

int Mpls_decodeLdpHashTlv
(mplsLdpHashTlv_t * hashTlv, u_char * buff, int bufSize) {
    u_char *hashTlvPtr;

    if (MPLS_HASHTLV_FIXLEN > bufSize) {

```



```

        PRINT_ERR("failed decoding hash tlv\n");
        return MPLS_DEC_BUFFTOOSMALL;
    }
    hashTlvPtr = (u_char *) hashTlv;
    hashTlvPtr += MPLS_TLVFIXLEN;

    MEM_COPY(hashTlvPtr, buff, MPLS_HASHTLV_FIXLEN);

    return MPLS_HASHTLV_FIXLEN;
}
/* End: Mpls_decodeLdpHashTlv */

```

As funções para a impressão das mensagens Label Mapping e Label Request foram alteradas para a imprimir os novos TLVs de Nonce e Hash caso existam (hashTlvExists e nonceTlvExists atribuídos em 1). Estas funções servem para fornecer um log da execução do LDP:

```

void printLlbMapMsg(ldp_instance_handle handle, mplsLdpLblMapMsg_t *
lblMapMsg)
void printLlbReqMsg(ldp_instance_handle handle, mplsLdpLblReqMsg_t *
lblReqMsg)

```

Foram criadas as funções "printNonceTlv" e "printHashTlv" para imprimir o conteúdo dos TLVs de Nonce e Hash (para finalidades de *debug*):

```

void printNonceTlv(ldp_instance_handle handle, mplsLdpNonceTlv_t * tlv)
void printHashTlv(ldp_instance_handle handle, mplsLdpHashTlv_t * tlv)

```

c) alterações efetuadas no arquivo "ldp-portable/lib/ldp_struct.h":

O arquivo ldp_struct.h contém as estruturas de dados manipuladas pelo LDP em tempo execução. Foi alterada a estrutura "ldp_return_enum" para inserir um novo estado LDP_AUTH_FAILURE usado nas mensagens LDP Notification para notificar que houve uma falha na autenticação:

```

typedef enum {
    LDP_SUCCESS = 1,
    ...
    // BEGIN : AUTH
    LDP_AUTH_FAILURE
    // END : AUTH
} ldp_return_enum;

```

Foi alterada a estrutura "ldp_notif_status" para inserir um novo "código de status" (status code) com o valor LDP_NOTIF_AUTH_FAILED ao TLV de Status do LDP. Este "código de status" deve ser retornado ao LSR origem caso uma tentativa de autenticação fim a fim venha a falhar:

```

typedef enum {
    LDP_NOTIF_NONE = 0,
    // BEGIN : AUTH
    LDP_NOTIF_AUTH_FAILED
    // END : AUTH
} ldp_notif_status;

```

Foi alterada a estrutura `ldp_attr`. Esta estrutura pode ser uma mensagem genérica ou uma FEC.

```
typedef struct ldp_attr {
    ...
    // BEGIN: AUTH
    mplsLdpNonceTlv_t nonceTlv;
    mplsLdpHashTlv_t hashTlv;
    // END: AUTH
    ...
    // BEGIN: AUTH
    uint8_t nonceTlvExists:1;
    uint8_t hashTlvExists:1;
    // END: AUTH
    ...
} ldp_attr;
```

d) alterações efetuadas no arquivo "ldp-portable/lib/ldp_pdu_setup.h":

Foram inseridos no arquivo `ldp_pdu_setup.h` os protótipos para as funções relacionadas a codificação dos TLVs de Nonce e Hash implementadas no arquivo `ldp-portable/lib/ldp_pdu_setup.c`.

```
// BEGIN: AUTH
int setupNonceTlv(mplsLdpNonceTlv_t * nonceTlv);
int setupHashTlv(mplsLdpHashTlv_t * hashTlv, mplsLdpNonceTlv_t * nonceTlv,
                mplsLdpFecTlv_t * fecTlv, u_int lsrid);
// END: AUTH
```

e) alterações efetuadas no arquivo "ldp-portable/lib/ldp_pdu_setup.c":

Foram inseridos os *includes* de arquivos necessários para autenticação fim a fim implementada. Os arquivos `ldp_label_mapping.c`, `ldp_label_request.c` e `ldp_auth.c`, também possuem estes mesmos includes:

```
// BEGIN: AUTH
#include <openssl/sha.h>
#include <openssl/rsa.h>
#include <openssl/objects.h>
#include <sys/time.h>
#include <stdio.h>
#include "ldp_auth.h"
// END: AUTH
```

Foi inserida a função para codificação (preenchimento de valores nos campos) do TLV de Nonce. O valor de nonce é gerado com a função `time` que retorna a hora atual em formato timestamp:

```
//BEGIN: AUTH
int setupNonceTlv(mplsLdpNonceTlv_t * nonceTlv)
{
    u_int nonce = time(NULL);
    nonceTlv->baseTlv.flags.flags.tBit = MPLS_NONCE_TLVTYPE;
    nonceTlv->baseTlv.flags.flags.uBit = 1;
    nonceTlv->baseTlv.flags.flags.fBit = 1;
    nonceTlv->baseTlv.length = MPLS_NONCETLV_FIXLEN;
    nonceTlv->nonce = nonce;
    return MPLS_NONCETLV_FIXLEN + MPLS_TLVFIXLEN;
}
```

```

    printf("EXIT: LDP_NONCE_TLV\n");
}
//END: AUTH

```

Foi inserida a função para codificação (preenchimento de valores nos campos) do TLV de Hash. É nesta função que o *hash* SHA1 é gerado, e o seu resultado e depois é encriptado (assinado) aplicando a função RSA com chave de 128 bits:

```

// BEGIN: AUTH
int setupHashTlv(mplsLdpHashTlv_t * hashTlv, mplsLdpNonceTlv_t * nonceTlv,
                mplsLdpFecTlv_t * fecTlv, u_int lsrAddress)
{
    unsigned char *hashDigest, *rsaSign;
    u_int i, signLen, labelSpace = 0;
    RSA *rsa;
    SHA_CTX sha;
    LDP_ENTER(NULL, "setupHashTlv");
    rsa = keyList_getPrivateKey();
    LDP_PRINT(NULL, "setupHashTlv: rsa = %p", rsa);
    rsaSign = malloc(RSA_size(rsa));
    hashDigest = malloc(SHA_DIGEST_LENGTH);
    SHA1_Init(&sha);
    SHA1_Update(&sha, &nonceTlv->nonce, sizeof(u_int));
    SHA1_Update(&sha, &(fecTlv->fecElArray[0].addressEl.address),
                sizeof(u_int));
    SHA1_Update(&sha, &(fecTlv->fecElArray[0].addressEl.preLen),
                sizeof(u_int));
    SHA1_Update(&sha, &lsrAddress, sizeof(u_int));
    SHA1_Final(hashDigest, &sha);
    LDP_PRINT(NULL, "setupHashTlv: nonce = %u, fec = %X/%d, lsrid = %X",
                nonceTlv->nonce, fecTlv->fecElArray[0].addressEl.address,
                fecTlv->fecElArray[0].addressEl.preLen, lsrAddress);
    LDP_PRINT(NULL, "setupHashTlv: hash = %s", hashDigest);
    RSA_sign(NID_sha1, hashDigest, SHA_DIGEST_LENGTH, rsaSign, &signLen,
            rsa);
    LDP_PRINT(NULL, "setupHashTlv: signLen = %d", signLen);
    LDP_PRINT(NULL, "setupHashTlv: sign = %s", rsaSign);
    hashTlv->baseTlv.flags.flags.tBit = MPLS_HASH_TLVTYPE;
    hashTlv->baseTlv.flags.flags.uBit = 1;
    hashTlv->baseTlv.flags.flags.fBit = 1;
    hashTlv->baseTlv.length = MPLS_HASH_TLV_FIXLEN;
    hashTlv->lsrAddress = lsrAddress;
    hashTlv->labelSpace = labelSpace;
    for(i = 0; i < signLen; i++)
        hashTlv->hashDigest[i] = rsaSign[i];
    LDP_EXIT(NULL, "setupHashTlv");
    return MPLS_HASH_TLV_FIXLEN + MPLS_TLV_FIXLEN;
}
// END: AUTH

```

Foi modificada a função de codificação do TLV de Status de modo a atribuir o valor do campo "unknow-bit" e "forward-bit" em 1, em casos de notificação de falha de autenticação, conforme definido pela proposta de autenticação fim a fim..

```

int setupStatusTlv(mplsLdpStatusTlv_t * statTlv, int fatal, int forward,
                  int status, unsigned int msgId, int msgType)
{
    ...

```

```

//BEGIN: AUTH
if (status == LDP_NOTIF_AUTH_FAILED){
    statTlv->baseTlv.flags.flags.uBit = 1;
    statTlv->baseTlv.flags.flags.fBit = 1;
}
//END: AUTH
...
}

```

f) alterações efetuadas no arquivo "ldp-portable/lib/ldp_label_mapping.c":

Foi modificada a função que transforma uma mensagem Label Mapping em uma estrutura ldp_attr:

```

void map2attr(mplsLdpLblMapMsg_t * map, ldp_attr * attr, uint32_t flag)
{
    ...
    // BEGIN: AUTH
    if (map->nonceTlvExists && flag & LDP_ATTR_NONCE) {
        memcpy(&attr->nonceTlv, &map->nonceTlv, sizeof(mplsLdpNonceTlv_t));
        attr->nonceTlvExists = 1;
    }
    if (map->hashTlvExists && flag & LDP_ATTR_HASH) {
        memcpy(&attr->hashTlv, &map->hashTlv, sizeof(mplsLdpHashTlv_t));
        attr->hashTlvExists = 1;
    }
    // END: AUTH
}

```

Foi modificada a função "ldp_label_mapping_initial_callback", que trata do envio mensagens Label Mapping de modo que sejam inseridos os TLVs de Nonce e Hash quando o LSR for o EGRESSO para a FEC sendo processada:

```

void ldp_label_mapping_initial_callback(ldp_timer_handle timer, void
*extra, ldp_cfg_handle g)
{
    ...
    // BEGIN: AUTH
    ldp_attr dummy;
    // END: AUTH
    ...
    if (g->lsp_control_mode == LDP_CONTROL_ORDERED) {
        if (ldp_policy_egress_check(g->user_data, &dest, s) == LDP_TRUE) {
            // BEGIN: AUTH
            if(auth_required()){
                memset(&dummy, 0, sizeof(ldp_attr));
                ldp_fec2fec_tlv(&fec, &dummy.fecTlv, 0);
                dummy.fecTlvExists = 1;
                dummy.fecTlv.numberFecElements = 1;
                dummy.nonceTlvExists = 1;
                dummy.hashTlvExists = 1;
                us_attr = &dummy;
            }
            // END: AUTH
            ...
        }
    }
}

ldp_return_enum ldp_label_mapping_send(ldp_global * g, ldp_session * s,
ldp_attr * us_attr, ldp_attr * ds_attr)
{
    ...
}

```

```

//BEGIN: AUTH
ldp_label_mapping_prepare_msg(msg, g->message_identifler++, us_attr, \
g->lsr_identifler);
//END: AUTH
...
}

```

Foi modificada a função `ldp_label_mapping_prepare_msg` para chamar a função de *setup* dos TLV's de Nonce e Hash:

```

void ldp_label_mapping_prepare_msg(ldp_msg * msg, uint32_t msgid,
ldp_attr * s_attr)
{
...
//BEGIN: AUTH
if (s_attr->nonceTlvExists) {
map->nonceTlvExists = 1;
map->baseMsg.msgLength += setupNonceTlv(&map->nonceTlv);
}
if (s_attr->hashTlvExists) {
map->hashTlvExists = 1;
map->baseMsg.msgLength += setupHashTlv(&map->hashTlv, \
&map->nonceTlv, &map->fecTlv, lsrAddress.u.ipv4);
}
// END: AUTH
...
}

```

Foi modificada a função "`ldp_label_mapping_process`", que trata do recebimento de mensagens Label Mapping, para processar a autenticação fim a fim proposta caso o LSR seja o INGRESSO para a FEC da mensagem. Se a autenticação falhar deve ser enviada uma mensagem LDP Notification com código de Status "Authentication Failed" ao LSR origem.

Para verificar se a autenticação é válida é gerado um hash dos valores recebidos pelo LSR, o qual é comparado com o valor do campo "Hash Digest" do TLV de Hash recebido, é verificada a assinatura da origem (função `RSA_verify`) e o valor de nonce é comparado com um *timestamp* local de modo a verificar se o tempo de validade do pacotes (10 minutos) não expirou.

```

ldp_return_enum ldp_label_mapping_process(ldp_global * g, ldp_session *
s, ldp_adj * a, ldp_entity * e, ldp_attr * r_attr, ldp_fec * fec)
...
if (ldp_mpls_outlabel_add(g->mpls_handle, out) != LDP_SUCCESS) {
// BEGIN: AUTH
Lmp_15:
// END: AUTH
...
}
// BEGIN: AUTH
if (ldp_policy_ingress_check(g->user_data, fec, nh_addr, nh_session) ==
LDP_TRUE) {
if (r_attr->hashTlvExists) {
unsigned char *hashDigest;
RSA* rsa;
SHA_CTX sha;

```

```

        LDP_ENTER(g->user_data,
"ldp_authentication_label_mapping_process");
        LDP_PRINT(g->user_data, "lsr id = %X", s->adj-
>remote_lsr_address.u.ipv4);

        rsa = keyList_findKey(s->adj->remote_lsr_address.u.ipv4);

        // checks if the received signature is OK
        if(rsa != NULL){
            //generates the hash of received tlv's and verifies the
signature
            hashDigest = malloc(SHA_DIGEST_LENGTH);
            SHA1_Init(&sha);
            SHA1_Update(&sha, &r_attr->nonceTlv.nonce, sizeof(u_int));
            SHA1_Update(&sha, &(r_attr-
>fecTlv.fecElArray[0].addressEl.address), sizeof(u_int));
            SHA1_Update(&sha, &(r_attr-
>fecTlv.fecElArray[0].addressEl.preLen), sizeof(u_int));
            SHA1_Update(&sha, &r_attr->hashTlv.lsrAddress, sizeof(u_int));
            SHA1_Final(hashDigest, &sha);

LDP_PRINT(g->user_data, "nonce = %u, fec = %08X/%d, lsrid = %X", r_attr-
>nonceTlv.nonce, r_attr->fecTlv.fecElArray[0].addressEl.address, r_attr-
>fecTlv.fecElArray[0].addressEl.preLen, r_attr->hashTlv.lsrAddress);
            LDP_PRINT(g->user_data, "hash = %s", hashDigest);
            LDP_PRINT(g->user_data, "hash = %s", r_attr-
>hashTlv.hashDigest);

            if( !RSA_verify(NID_sha1, hashDigest, SHA_DIGEST_LENGTH, r_attr-
>hashTlv.hashDigest, 128, rsa) ) {
                LDP_TRACE_LOG(g->user_data, LDP_TRACE_STATE_RECV,
LDP_TRACE_FLAG_LABEL, "LDP Authentication failed for LSR %08x:%d FEC
%08x/%d\n", s->adj->remote_lsr_address.u.ipv4,
s->adj->remote_label_space,
r_attr->fecTlv.fecElArray[0].addressEl.address,
r_attr->fecTlv.fecElArray[0].addressEl.preLen);

                ldp_notif_send(g, s, r_attr, LDP_NOTIF_AUTH_FAILED);
                LDP_EXIT(g->user_data, "ldp_authentication_process");
                goto Lmp_15; //release all labels if auth faill
            }
            else {
                // check if nonce tlv is still valid
                time_t abstime = time(NULL);
                struct tm *ltime, *rtime;
                ltime = gmtime(&abstime);
                rtime = gmtime(&r_attr->nonceTlv.nonce);
                if(ltime->tm_min > rtime->tm_min + 10){
                    LDP_PRINT(g->user_data, "Nonce TLV expitred for LSR %08x:%d
FEC %08x/%d\n", s->adj->remote_lsr_address.u.ipv4,
s->adj->remote_label_space,
r_attr->fecTlv.fecElArray[0].addressEl.address,
r_attr->fecTlv.fecElArray[0].addressEl.preLen);
                    ldp_notif_send(g, s, r_attr, LDP_NOTIF_AUTH_FAILED);
                    LDP_EXIT(g->user_data, "ldp_authentication_process");
                }
                else {
                    LDP_PRINT(g->user_data, "LDP Authentication Success for LSR
%08x:%d FEC %08x/%d\n", s->adj->remote_lsr_address.u.ipv4,
s->adj->remote_label_space,
r_attr->fecTlv.fecElArray[0].addressEl.address,
r_attr->fecTlv.fecElArray[0].addressEl.preLen);
                    LDP_EXIT(g->user_data, "ldp_authentication_process");
                }
            }
        }
        // no authentication is required if we don't know the LSR
        // proceed with the mapping
    }
}

```

```

        else {
            LDP_TRACE_LOG(g->user_data, LDP_TRACE_STATE_RECV,
LDP_TRACE_FLAG_LABEL, "LDP No Authentication Required for LSR %08x:%d FEC
%08x/%d\n", s->adj->remote_lsr_address.u.ipv4,
s->adj->remote_label_space,
r_attr->fecTlv.fecElArray[0].addressEl.address,
r_attr->fecTlv.fecElArray[0].addressEl.preLen);
            LDP_EXIT(g->user_data,
"ldp_authentication_label_mapping_process");
        };
    }
// END: AUTH

```

g) alterações efetuadas no arquivo "ldp-portable/lib/ldp_label_request.c:

Neste arquivo foram efetuadas as alterações necessárias definidas pela autenticação fim a fim proposta e a compilação do mesmo não apresentou erros. Não foi possível testar o funcionamento das mensagens Label Request, pois o ldp-portable, na versão utilizada por esta implementação (versão 0.200), fica instável quando configurado para operar no modo de Distribuição SobDemanda.

Na distribuição SobDemanda um LSR só distribui rótulos para seus pares LDP se estes solicitarem este procedimento, e justamente para realizar estas solicitações de rótulos que são empregadas mensagens Label Request. Por este motivo é necessário o uso do modo de Distribuição SobDemanda para testar o uso de mensagens Label Request.

As alterações efetuadas neste arquivo não foram listadas, porém podem ser conferidas no código fonte desta implementação.

h) alterações efetuadas no arquivo "ldp-portable/lib/ldp_state_machine.c":

Foi alterada a função ldp_event para notificar falha na autenticação fim a fim:

```

ldp_return_enum ldp_event(ldp_cfg_handle g, ldp_socket_handle socket,
    ldp_dest * user_from, ldp_if_handle if_handle, ldp_buf * buf, void
*extra,
    ldp_event_enum event)
{
    ...
    //BEGIN: AUTH
    case LDP_AUTH_FAILURE:
    {
        LDP_TRACE_LOG(g->user_data, LDP_TRACE_STATE_ALL,
LDP_TRACE_FLAG_ERROR,
        "ldp_event: LDP AUTH FAILURE\n");
        break;
    }
    //END: AUTH
    ...
}

```

i) alterações efetuadas no arquivo "ldp-portable/lib/ldp_notif.c":

Foi modificada a função "ldp_notif_prepare_msg" para atribuir o campo "forward-bit" em 1 no TLV de Status do LDP quando a autenticação falhar:

```
void ldp_notif_prepare_msg(ldp_mesg * msg, uint32_t msgid, ldp_attr *
r_attr,
    ldp_notif_status status)
{
    ...
    if (status == LDP_NOTIF_LOOP_DETECTED ||
        status == LDP_NOTIF_UNKNOWN_FEC ||
        status == LDP_NOTIF_NO_ROUTE
        //BEGIN: AUTH
        || status == LDP_NOTIF_AUTH_FAILED)
        //END: AUTH
        {
            forward = 1;
        } else {
            forward = 0;
        }
    ...
}
```

Foi modificada a função "ldp_notif_process" para retornar a constante LDP_AUTH_FAILURE e imprimir a mensagem de falha de autenticação:

```
ldp_return_enum ldp_notif_process(ldp_global * g, ldp_session * s,
    ldp_adj * a, ldp_entity * e, ldp_attr * r_attr)
{
    ...
    //BEGIN : AUTH
    case LDP_NOTIF_AUTH_FAILED:
        retval = LDP_AUTH_FAILURE;
        fprintf(stderr, "AUTHENTICATION FAILED, status: %08x\n", status);
        break;
    //END : AUTH
    ...
}
```

j) arquivo "ldp-portable/lib/ldp_auth.h" criado pela implementação da autenticação:

O arquivo ldp_auth.h foi criado pela autenticação fim a fim. Este arquivo contém as constantes usadas na autenticação; define uma estrutura (pubKeyElement) composta pelo LSR-ID e a chave pública dos LSRs que precisam se autenticar frente ao LSR e define um lista encadeada para armazenamento destas estruturas e protótipos usados pelas funções existentes no arquivo "ldp-portable/lib/ldp_auth.c":

```
// BEGIN: AUTH
#ifdef ldp_auth_H
#define ldp_auth_H

#include <openssl/rsa.h>

//Constantes de autenticação
#define LDP_AUTH_PRIVATE_KEY_FILE "/usr/local/etc/ldp_key"
```



```

#define LDP_AUTH_PUB_KEY_FILE                "/usr/local/etc/ldp_key.pub"
#define LDP_AUTH_PEERS_KEY_FILE            "/usr/local/etc/ldp_peers"

//elemento chave/lsr-id
typedef struct{
    unsigned long lsr_id;
    RSA* key;
} pubKeyElement_t;

//lista de elementos pubKeyElement
struct keyListNode_s{
    pubKeyElement_t *data;
    struct keyListNode_s *next;
};

typedef struct keyListNode_s keyListNode_t;

typedef struct{
    keyListNode_t *first, *last;
    int size;
} keyList_head_t;

//protótipos de funções do ldp_auth.c
int auth_init();
void keyList_cleanup();
int keyList_readFrom(char *from);
int auth_required();
RSA *keyList_findKey(u_int id);
RSA *keyList_getPrivateKey();
RSA *keyList_getPublicKey();

#endif
// END: AUTH

```

k) arquivo "ldp-portable/lib/ldp_auth.c" criado pela implementação da autenticação:

O arquivo ldp_auth.c foi criado pela autenticação fim a fim. Este arquivo contém as principais funções usadas pela autenticação fim a fim. Foram inseridos os arquivos de include necessários para a autenticação fim a fim:

```

// BEGIN AUTH
#include <openssl/rsa.h>
#include <openssl/sha.h>
#include <openssl/pem.h>
#include <openssl/objects.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include "ldp_auth.h"

```

Foram definidas funções para ler a chave pública e privada do LSR e a lista de LSRs que precisam se autentica frente a este LSR:

```

static keyList_head_t keyList_head = {NULL, NULL, 0};
static RSA *privateKey = NULL, *publicKey = NULL;

int auth_init(){
    FILE* fp;
    int ret;

```

```

privateKey = NULL;
publicKey = NULL;

printf("ENTER: auth_init\n");

OpenSSL_add_all_algorithms();

printf("auth_init: priv. file = %s\n", LDP_AUTH_PRIVATE_KEY_FILE);

// reads this lsr's private key
fp = fopen(LDP_AUTH_PRIVATE_KEY_FILE, "r");
printf("auth_init: priv. file = %p\n", fp);
if(fp == NULL) return -1;
privateKey = PEM_read_RSAPrivateKey(fp, &privateKey, NULL, "LDP_AUTH");
printf("auth_init: priv. key = %p\n", privateKey);
if(privateKey == NULL) return -3;
fclose(fp);

// reads this lsr's public key
fp = fopen(LDP_AUTH_PUB_KEY_FILE, "r");
printf("auth_init: pub. file = %s\n", LDP_AUTH_PUB_KEY_FILE);
printf("auth_init: pub. file = %p\n", fp);
if(fp == NULL) return -2;
publicKey = PEM_read_RSAPublicKey(fp, &privateKey, NULL, NULL);
printf("auth_init: pub. key = %p\n", publicKey);
if(publicKey == NULL) return -4;
fclose(fp);

// creates list of lsrs that must authenticate
printf("auth_init: peers file = %s\n", LDP_AUTH_PEERS_KEY_FILE);
ret = keyList_readFrom(LDP_AUTH_PEERS_KEY_FILE);
printf("auth_init: peers ret = %d\n", ret);

if(ret < 0) return -5;

return 1;
};

```

Foi definida uma função para limpar a lista de chaves em memória:

```

void keyList_cleanup(){
    keyListNode_t *aux, *current = keyList_head.first;
    while(current){
        RSA_free(current->data->key);
        free(current->data);
        aux = current;
        current = current->next;
        free(aux);
    };

    keyList_head.size = 0;
};

```

Foi definida uma rotina para ler o arquivo de chaves passo a passo e armazenar em uma lista encadeada:

```

int keyList_readFrom(char *from){
    FILE* fp = fopen(from, "r");
    char buf[500];
    int r;
    struct in_addr ia;
    keyListNode_t **current;

    if(fp == NULL) return -1;

```

```

current = &keyList_head.first;

    keyList_head.size = 0;
    while(fgets(buf, 500, fp)){
        r = inet_aton(buf, &ia);
        (*current) = malloc(sizeof( keyListNode_t));
        (*current)->data = malloc(sizeof(pubKeyElement_t));
            (*current)->data->lsr_id = ntohl(ia.s_addr);
        (*current)->data->key = NULL;
        (*current)->data->key = PEM_read_RSAPublicKey(fp,
            &(*current)->data->key, NULL, NULL);
printf("ip = %s, id = %X\n", buf, (*current)->data->lsr_id);
        if((*current)->data->key == NULL || r == 0)
            return -1;

        (*current)->next = NULL;
        keyList_head.last = *current;
        current = &((*current)->next);
        keyList_head.size++;
    };
*current = NULL;
    return keyList_head.size;
};

```

Foi definida uma rotina para procurar uma determinada chave na lista encadeada:

```

RSA *keyList_findKey(u_int id){
    keyListNode_t *current = keyList_head.first;

    while(current){
        if(current->data->lsr_id == id)
            return current->data->key;
        current = current->next;
    };

    return NULL;
};

//Retorna a chave privada do lsr
RSA *keyList_getPrivateKey(){
    return privateKey;
};

//Retornar a chave pública do lsr
RSA *keyList_getPublicKey(){
    return publicKey;
};

//Verifica se o LSR está configurado para utilizar a autenticação fim-a-
fim
int auth_required(){
    if(privateKey)
        return 1;
    else
        return 0;
};
// END: AUTH

```

Além dos arquivos de biblioteca do ldp-portable, localizados no diretório "ldp-portable/lib", foram alterados dois outros arquivos da distribuição do zebra, são eles: zebra/mpls/mpls.c e zebra/mpls/Makefile:

l) alterações efetuadas no arquivo "zebra/mpls/mpls.c":

O arquivo mpls.c é o fonte do arquivo executável "mplsd". Nele foram feitas alterações para inicializar os procedimentos responsáveis pela autenticação implementada. As alterações foram incluir o arquivo de protótipo das funções de autenticação:

```
//BEGIN: AUTH
#include "ldp_auth.h"
//END: AUTH
```

Chamar a função que inicializa os procedimentos da autenticação, como inicializar variáveis e fazer verificações de arquivos e configurações necessárias para o funcionamento do ambiente:

```
//BEGIN: AUTH
auth_init();
//END: AUTH
```

m) alterações efetuadas no arquivo "zebra/mpls/Makefile":

O arquivo Makefile utilizado para compilar a distribuição do zebra foi modificado para o incluir os arquivos criados pela implementação da autenticação fim a fim:

```
// BEGIN: AUTH
mplsd_SOURCES = ... ldp_auth.c
noinst_HEADERS = ... ldp_auth.h
am_mpls_OBJECTS = ... ldp_auth.$(OBJEXT)
LIBS = ... -lssl
// END: AUTH
```

2.4.2 - Validação da proposta

Para validar a proposta foi montado um ambiente de testes utilizando computadores rodando linux com suporte à MPLS e LDP.

O LDP-PORTABLE é uma plataforma de código fonte aberto (*open source*) e ainda está em fase inicial de desenvolvimento. Dessa forma muitas funcionalidades do protocolo LDP definidas na RFC 3036 ainda não estão implementadas ou não funcionam completamente.

Foi necessário compilar os fontes dos programas utilizados e habilitar suporte ao MPLS/LDP no kernel do linux. Os programas necessários para compilar o linux com suporte a MPLS foram:

- linux kernel 2.4.19, disponível por <ftp://ftp.kernel.org/pub/linux/kernel/v2.4/linux-2.4.19.tar.gz>.

- mpls-linux-1.170, disponível por: <http://prdownloads.sourceforge.net/mpls-linux/mpls-linux-1.170.tar.gz?download>.

Aplicar o mpls-linux (*patch*) ao kernel do linux de modo a fornecer suporte a MPLS.

Para a compilação da ferramenta zebra com o módulo de LDP, os programas necessários foram:

- ldp-portable-0200, disponível por: <http://prdownloads.sourceforge.net/mpls-linux/ldp-portable-0.200.tar.gz?download>
- zebra-0.93. O download da versão do zebra foi realizado diretamente da área de desenvolvimento com os comandos abaixo:

```
# CVSROOT=:pserver:anoncvs@anoncvs.zebra.org:/cvsroot";
# export CVSROOT; cvs login
# cd /usr/src
# cvs -z3 co -D "Dom Sep 1 00:00:00 2002" zebra
```

- automake-1.6.2, disponível por: <ftp://ftp.gnu.org/gnu/autoconf/automake-1.6.2.tar.bz2>.
- autoconf-2.5.3, disponível por: <ftp://ftp.gnu.org/gnu/autoconf/autoconf-2.53.tar.bz2>.

Depois de obter as ferramentas, foi executada a seguinte sequência de passos para a instalação do ambiente de testes usando a distribuição de linux REDHAT-7.1.

Passo 1: extrair o kernel do linux;

Passo 2: instalar o pacote mpls-linux;

Passo 3: habilitar suporte ao MPLS/LDP no kernel e compilar;

Passo 4: extrair os fontes do ldp-portable e do zebra;

Passo 5: instalar o pacote do ldp-portable ao zebra e compilar.

Como resultado da compilação do zebra foram gerados os binários "mplsd" e "zebra" os quais ficam armazenados por default em /usr/local/sbin.

2.4.3 - Configuração e Execução no Ambiente de Testes

Foi configurado um ambiente de testes em duas máquinas Linux nomeadas como LERA e LERB respectivamente, para verificar o funcionamento da autenticação fim a fim implementada ao LDP. O ambiente de testes é mostrado na figura 25.

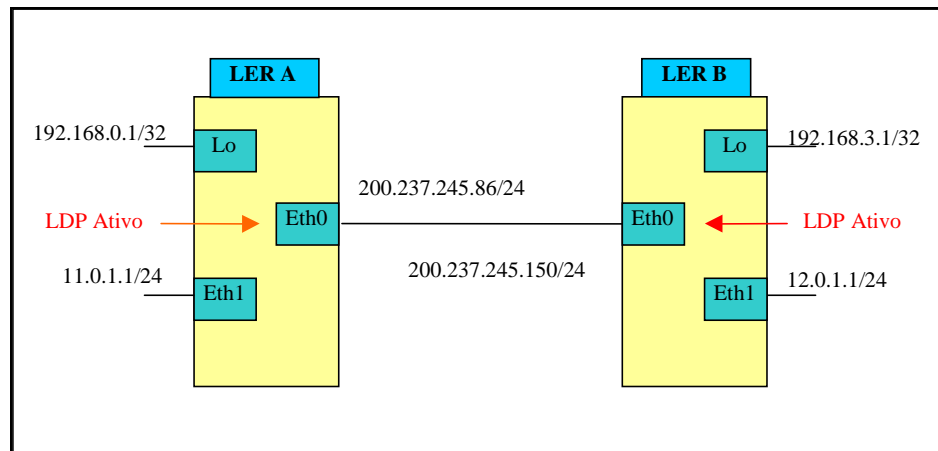


Figura 25 - Ambiente de testes

A figura 25 mostra o IP atribuído a cada interface de rede neste ambiente de testes. Por exemplo, o LERA, na interface "loopback" (Lo) possui o IP 192.168.0.1 com máscara de rede (/32) que equivale a representação decimal 255.255.255.255. As máquinas estão conectadas através do barramento "Ethernet0" (Eth0).

Foram configuradas rotas estáticas em cada LSR da seguinte forma:

```
LER A:
route add 192.168.3.1/32 gw 200.237.245.150
route add 12.0.0.0/8 gw 200.237.245.150
*ldp enabled on eth0
```

```
LER B:
route add 192.168.0.1/32 gw 200.237.245.86
route add 11.0.0.0/8 gw 200.237.245.86
*ldp enabled on eth0
```

Para iniciar a execução do LDP, ou seja, a troca de mensagens LDP para gerência de rótulos MPLS entre o LERA e o LERB, foi necessário criar um arquivo de configuração em cada LSR e executar alguns comandos básicos de inicialização do ambiente. No ANEXO 2 podem ser encontrados os passos detalhados para configurar e executar o LDP em cada um dos LSRs (LERA e LERB).

2.4.3.1 - Interface de Gerenciamento

A interface do "mplsd" fornece alguns comandos para visualizar o ambiente em execução. É necessário analisar o resultado destes comandos em cada LSR do ambiente para se ter uma visão completa das negociações realizadas pelo LDP no ambiente.

Os comandos abaixo foram executados no LERA, durante a execução do ambiente de testes como forma de exemplificar os resultados exibidos pelos comandos de visualização do ambiente LDP em execução.

```
# telnet localhost 2603
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is zebra (version 0.93a).
Copyright 1996-2002 Kunihiro Ishiguro.

User Access Verification
Password:
LerA#
```

O comando abaixo mostra a configuração global do LDP no LERA.

```
LerA# show ldp
LSR-ID: c0a80001 Admin State: ENABLED
Transport Address: 00000000
Control Mode: ORDERED Repair Mode: GLOBAL
Propagate Release: TRUE Label Merge: TRUE
Retention Mode: LIBERAL Loop Detection Mode: NONE
TTL-less-domain: FALSE
Local TCP Port: 646 Local UDP Port: 646
Keep-alive Time: 45 Keep-alive Interval: 15
Hello Time: 15 Hello Interval: 5
```

O próximo comando mostra os pares LDP (vizinhos) que o LERA localizou, neste caso apenas o LERB, e as informações LDP referentes a este par.

```
LerA_ip86# show ldp neighbors
Peer LDP Ident: 192.168.3.1:0; Local LDP Ident: 192.168.0.1:0
TCP connection: n/a
State: OPERATIONAL; Msgs sent/recv: 10/8; UNSOLICITED
Up time: n/a
LDP discovery sources:
eth0
Addresses bound to peer:
200.237.245.150 192.168.3.1 12.0.1.1
```

O próximo comando mostra as sessões que o LERA mantém com seus pares, neste caso com o LERB.

```
LerA_ip86# show ldp session
1 200.237.245.150 45 OPERATIONAL
200.237.245.150
192.168.3.1
12.0.1.1
```

O próximo comando mostra as atribuições de rótulo(*label*)/FEC que o LERA recebeu dos seus pares LDP, neste caso do LERB (via mensagens Label Mapping). O comando exibe a FEC em questão (192.168.3.1/32), o label de saída (label: gen 16), e quem é próximo hop, expresso pelo LSR-ID + espaço de *labels* (lsr: 192.168.3.1:0) do LSR remoto.

```
LerA_ip86# show ldp database
192.168.3.1/32 remote binding: label: gen 16 lsr: 192.168.3.1:0
installed
```

Para visualizar os LSPs estabelecidos no ambiente é necessário acessar a interface do "zebra", em ambos os LSRs (LERA e LERB), e executar o comando abaixo.

```
# telnet localhost 2603
Trying 127.0.0.1 ...
...
Zebra_LerA#
```

O comando abaixo mostra as rotas ip do LERA e também os LSPs estabelecidos via MPLS (repare a linha em negrito na saída do comando abaixo).

EM relação ao LSP está sendo listando o prefixo de endereços IP, ou FEC, (192.168.3.1/32), o próximo *hop* (200.237.245.150) para esta FEC, a interface física de saída (eth0) e o flag "MPLS" (expressa que se trata de um LSP) associado a um identificador do *label* de saída (0x2) utilizado pelo MPLS no kernel do linux. Como os LSPs são unidirecionais, no LERB existirá um LSP na direção oposta de modo a prover uma comunicação bidirecional via MPLS.

```
zebra_LERA> show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
      B - BGP, > - selected route, * - FIB route

K>* 0.0.0.0/0 via 200.237.245.254, eth0
C>* 11.0.1.0/24 is directly connected, eth1
K>* 12.0.0.0/8 via 200.237.245.150, eth0
C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.0.1/32 is directly connected, lo
K>* 192.168.3.1/32 via 200.237.245.150, eth0 MPLS 0x2
C>* 200.237.245.0/24 is directly connected, eth0
```

Para visualizar a tabela de *labels*, ou seja, a LIB (*Label Information Base*) do LERA é necessário executar o comando abaixo. Repare na linha em negrito, o identificador da LIB (**0x2**), listado pelo comando "show ip route" do zebra, referenciado no parágrafo anterior.

```
# cat /proc/net/mpls_*
/proc/net/mpls_in:
```



```
0x40004000 0/0/0 gen 16 0 1 POP PEEK
/proc/net/mpls_labelspace:
eth0      0      14
/proc/net/mpls_out:
0x00000002 0/0/0 1 PUSH(gen 16) SET(eth0,200.237.245.150)
```

2.4.4 - Resultados Obtidos

Na configuração *default*, o LDP-PORTABLE inicializa o LDP no Modo de Controle Ordenado e Distribuição NÃO SOLICITADA. Se o LDP-PORTABLE, na versão utilizada neste trabalho (versão 0.200), for configurado para operar no Modo de Distribuição SOB DEMANDA a versão fica instável e a execução é encerrada repentinamente. Como a versão está em desenvolvimento, não existe na interface de configuração do "mplsd" uma opção para mudar o Modo de Distribuição do LDP, apesar de ambos modos estarem implementados no código fonte. Para testes é necessário modificar o Modo de Distribuição diretamente no código fonte (*ldp-portable/lib/ldp_defaults.h*), habilitando a opção abaixo e após recompilar:

```
"#define LDP_ENTITY_DEF_DISTRIBUTION_MODE LDP_DISTRIBUTION_ONDEMAND"
```

Pelo fato do LDP ficar instável se configurado para operar no Modo de Distribuição Sob Demanda, esta implementação da proposta de autenticação foi testada apenas com o Modo de Distribuição NÃO SOLICITADO do LDP. O código necessário para utilizar a autenticação fim a fim no Modo de Distribuição SOB DEMANDA foi devidamente inserido e compilou sem erros, porém não foi possível testar seu funcionamento por causa da instabilidade de operação do LDP-PORTABLE neste modo de distribuição.

Referente a criação de LSPs, o "mplsd" se comporta da seguinte maneira: quando o LDP é iniciado, o mesmo processa a tabela de roteamento dessa máquina e tenta estabelecer os LSPs baseado nas informações da tabela de roteamento IP deste LSR, através de negociações com os pares LDP descobertos. Se a tabela de roteamento IP do LSR sofrer alterações, as mesmas serão refletidas nos LSPs criados. Dessa forma os LSPs não são criados baseados em fluxos de tráfego e sim baseados nas informações do nível IP (camada de rede) do LSR.

Os resultados foram obtidos no ambiente de testes demonstrado na figura 25.

Quando o LDP é inicializado em ambos os LSRs, LERA e LERB, as interfaces de rede de cada LSR são localmente identificadas e por *default* o "mplsd" assume como Identificador do LSR (LSR-ID) o endereço IP da interface de rede *loopback* (Lo), acrescido do espaço de *labels* associado a interface Lo, por default é 0, dessa forma:

- o LERA assume como LSR-ID o IP 192.168.0.1:0 (em Hexadecimal c0a80001:0);
- o LERB assume como LSR-ID o IP 192.168.3.1:0 (em Hexadecimal c0a80301:0).

Após esta etapa acontecem as negociações iniciais do protocolo LDP onde são executados os seguintes passos:

- troca de mensagens LDP HELLO: usadas para descobrir os vizinhos LDP no barramento local (adjacentes). O LERA descobre o LERB e vice-versa;
- troca de mensagens LDP INIT: usadas para estabelecer sessões LDP entre os vizinhos adjacentes, fazendo com que o LERA e o LERB tornem-se pares LDP. O LERB inicia o estabelecimento da sessão, pois possui o IP maior;
- troca de mensagens LDP KEEP ALIVE: usadas para manter as sessões LDP e as adjacências de HELLO entre o LERA e o LERB;
- troca de mensagens LDP ADDRESS: cada LSR (LERA e LERB) informa os endereços IP de suas interfaces de rede, na ordem (ethernet0, loopback, ethernet1), ao seu par LDP e recebe as mesmas informações deste.

Após estes passos começa a fase de troca de rótulos entre o LERA e o LERB.

O "mplsd" procura na tabela de roteamento do LSR uma entrada de *host* (prefixo=32) igual ao seu LSR-ID. Se conseguir satisfazer esta condição, inicia por esta FEC a troca de mensagens com os vizinhos LDP ativos.

A seguir serão apresentados os eventos de negociação de rótulos ocorridos no ambiente de testes. Conforme poderá ser constatado, apenas são utilizadas mensagens Label Mapping. Isso se deve ao fato do LDP estar configurado no Modo de Distribuição Não Solicitado, sendo assim, quando um LSR possui um novo *label* ele automaticamente atualiza seus vizinhos LDP com esta informação, logo não são empregadas mensagens Label Request. No Modo de Distribuição NÃO SOLICITADO o LDP pode utilizar mensagens Label Request, porém isso só ocorre em casos específicos

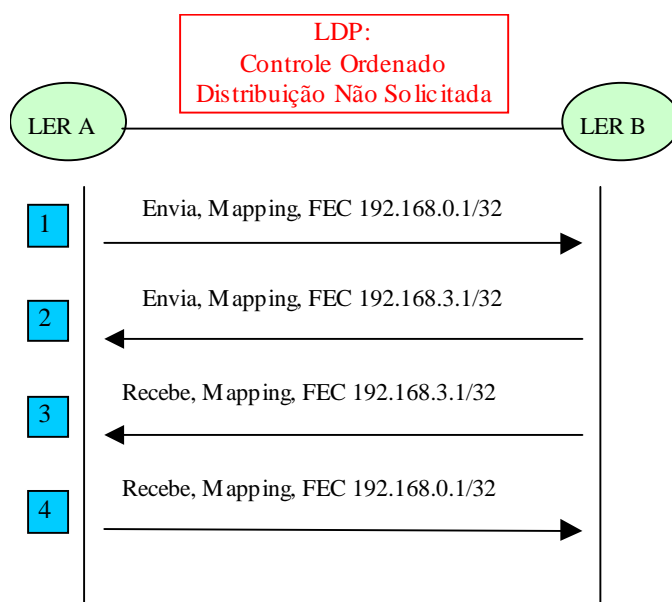


Figura 26 – Troca de mensagens entre os LSRs

O diagrama de seqüência apresentado na figura 26 mostra a seqüência dos eventos de troca de *labels* ocorridos durante a execução do ambiente de testes, os quais serão detalhados a seguir.

A autenticação fim a fim só é processada durante o recebimento de mensagens Label Request ou Label Mapping.

A figura 26 mostra a seqüência exata dos eventos ocorridos durante a execução do ambiente de testes (figura 25). Porém de forma a permitir uma melhor compreensão, os eventos de envio por parte do LERB e respectivo recebimento pelo LERA (2º e 3º eventos), quando a autenticação é processada pela primeira vez no ambiente; e os eventos de envio por parte do LERA e recebimento pelo LERB (1º e 4º eventos), quando a autenticação é processada pela segunda e última vez no ambiente, serão exibidos em seqüência.

2º evento) o LERB envia uma mensagem LDP Label Mapping para o LERA:

No 2º evento o LERB (c0a80301) envia uma mensagem LDP Label Mapping para o LERA (c0a80001) em relação a FEC 192.168.3.1/32 (em Hexadecimal c0a80301/32), que é também seu LSR-ID do LERB:

"OUT: Label Mapping Sent to c0a80001:0 for c0a80301/32".

Como o prefixo de rede IP 192.168.3.1/32 é local ao LERB (interface Lo), o LERB será o EGRESSO em relação a FEC 192.168.3.1/32. Por estar configurado no modo de distribuição NÃO SOLICITADO, onde o LSR anuncia

seus novos mapeamentos Etiqueta-FEC sem que ninguém os solicite, o LERB deve:

- gerar um *label* de entrada para esta FEC;
- codificar os TLVS da autenticação fim a fim (NonceTLV e HashTLV); e
- enviar uma mensagem LDP Label Mapping ao seu par LDP ativo (LERAB), de forma a criar um LSP com o LERA em relação a esta FEC;

Estes passos foram executados com sucesso conforme o bloco de *log* gerado pela execução do "mplsd" no ambiente de testes, abaixo apresentado (as linhas iniciadas por "//" são comentários em relação a saída de log subsequente).

```

----- Inicio do Bloco de LOG -----
...
// inicia a função que lê seqüencialmente a tabela de roteamento do LERB
ENTER: ldp_label_mapping_initial_callback
OUT: Initial Label Mapping fired: session(2)
ldp_label_mapping_with_xc: enter
ENTER: Prepare_Label_Mapping_Attributes
EXIT: Prepare_Label_Mapping_Attributes
ENTER: ldp_label_mapping_send
// label de entrada adicionado (valor do label = 16, conforme abaixo)
OUT: In Label Added
// rotina que gera a assinatura digital (hash encriptado com a chave
// privada do LSR 192.168.3.1)
ENTER: setupHashTlv
// valores de entrada para funcao hash sha1-160 bits
PRT: setupHashTlv: nonce = 1034717059, fec = C0A80301/32, lsrid = C0A80301
// resultado da função hash
PRT: setupHashTlv: hash = "E-™_ŷÆ],™3½t_“¶□CÔy%_
// assinatura RSA, com chave de 128 bits, aplicada sobre o valor hash
PRT: setupHashTlv: signLen = 128
PRT: setupHashTlv: sign =
ÊÊÊÊ_GkGîbés3fDã_2_Äæ<Ë,/N++B>ä°-1¿õâ__,Ilª#g%8Nip;²0iãjH-«²A
'ð°fBp<V-Ô6Ê¼+D@ñ,²ÔI}úAð'Š"nãªÖï
EXIT: setupHashTlv
// imprime campos e TLVs da mensagem Label Mapping que será enviada
OUT: LPD Header : protocolVersion = 1
OUT: pduLength = 180
// LSR de origem (LERB)
OUT: lsrAddress = c0a80301
OUT: labelSpace = 0
OUT: LABEL MAPPING MSG ***START***:
OUT: baseMsg : uBit = 0
OUT: msgType = 400
OUT: msgLength = 170
OUT: msgId = 7
OUT: fecTlv:
OUT: Tlv:
OUT: BaseTlv: uBit = 0
OUT: fBit = 0
OUT: type = 100
OUT: length = 8
OUT: fecTlv->numberFecElements = 1
OUT: elem 0 type is 2
// FEC e prefixo de rede (192.168.3.1/32)

```

```

OUT:          Fec Element : type = 2, addFam = 1, preLen = 32, address =
c0a80301
OUT:
OUT:      fecTlv.wcElemExists = 0
OUT:      genLblTlv:
OUT:      Tlv:
OUT:      BaseTlv: uBit = 0
OUT:          fBit = 0
OUT:          type = 200
OUT:          length = 4
// label atribuído como label de entrada para a FEC 192.168.3.1/32
OUT:      genLbl data: label = 16
OUT:      Label mapping msg does not have atm label Tlv
OUT:      Label mapping msg does not have fr label Tlv
OUT:      Label mapping msg does not have hop count Tlv
OUT:      Label mapping msg does not have path vector Tlv
OUT:      Label mapping msg does not have label messageId Tlv
OUT:      Label mapping msg does not have LSPID Tlv
OUT:      Label mapping msg does not have traffic Tlv
// imprime o 1º TLV da autenticação fim a fim (TLV de Nonce), o qual foi
// inserido porque o LERB é o LSR de EGRESSO para a FEC 192.168.3.1/32
OUT:      nonceTlv:
OUT:      Tlv:
OUT:      BaseTlv: uBit = 1
OUT:          fBit = 1
OUT:          type = 881
OUT:          length = 4
// timestamp gerado pelo LERB baseado na sua hora local
OUT:      nonceTlv data: Nonce = 1034717059
// imprime o 2º TLV da autenticação fim a fim (TLV de Hash), o qual foi
// inserido porque o LERB é o LSR de EGRESSO para a FEC 192.168.3.1/32
OUT:      hashTlv:
OUT:      Tlv:
OUT:      BaseTlv: uBit = 1
OUT:          fBit = 1
OUT:          type = 880
OUT:          length = 134
// Campos do TLV de Hash. (hashValue está encriptado com a chave
// privada do LERB)
OUT:      hashTlv data:
OUT:      lsrAddress = c0a80301, labelSpace = 0, hashDigest =
ÊÊÊÊš_GkGîbés3fDã_2_Äœè<Ë,/N++B>ä°-1çõâ__,Ilª#g%8Nip;²0iãjH-«²A
'ô°fBp<V-Ô6Ê¼+D@ñ,²ÔI}úAð'š"nãªÖi
OUT: LABEL MAPPING MSG ***END***:
// imprime mensagem confirmando envio da mensagem Label Mapping para
// o LERB em relação a FEC 192.168.3.1/32
OUT: Label Mapping Sent to c0a80001:0 for c0a80301/32
// encerra a função de envio de Label Mapping
EXIT: ldp_label_mapping_send
ldp_label_mapping_with_xc: exit
// encerra a função que leu sequencialmente a tabela de roteamento do LERB
EXIT: ldp_label_mapping_initial_callback
...
----- Fim do Bloco de LOG -----

```

Figura 27 -Envio de mensagem Label Mapping do LERB para o LERA

3º evento) o LERA recebe uma mensagem LDP Label Mapping do LERB e a autenticação é processada pela primeira vez no ambiente:

No 3º evento o LERA recebe uma mensagem LDP Label Mapping do LERB em relação a FEC 192.168.3.1/32 (em Hexadecimal c0a80301/32):

"OUT: Label Mapping Recv from c0a80301:0 for c0a80301/32".

Como existe uma entrada na tabela de roteamento do LERA indicando que o LERB é o próximo hop para a FEC 192.168.3.1/32, o LERA detecta que é o INGRESSO para esta FEC, dessa forma deve:

- adicionar o *label* recebido como *label* de saída para a FEC 192.168.3.1/32;
- processar os TLVS da autenticação fim a fim (NonceTLV e HashTLV) de modo a validar se o LERB tem autorização para estabelecer LSPs com o LERA; e
- estabelecer o LSP com o LERB, dessa forma o LSP terá o como INGRESSO, o LERA, e o como EGRESSO , o LERB. O sentido do LSP será do LERA em direção ao LERB. Lembre-se que os LSPs são unidirecionais, para ter uma conicação bidirecional entre dois LSRs é necessário criar um LSP no sentido inverso.

Estes passos foram executados com sucesso conforme o bloco de log gerado pela execução do "mplsd" no ambiente de testes, abaixo apresentado (as linhas iniciadas por "//" são comentários em relação a saída de *log* subsequente):

```
----- Inicio do Bloco de LOG -----
...
// novo evento detectado
ENTER: ldp_event
// inicia o processamento do buffer de entrada
ENTER: ldp_buf_process
// inicia decodificação do cabeçalho LDP e mostra o resultado
ENTER: ldp_decode_header
OUT: 00 01 00 b4 c0 a8 03 01 00 00
// encerra o processamento do cabeçalho LDP
EXIT: ldp_decode_header
// inicia decodificação do corpo da mensagem recebida
ENTER: ldp_decode_one_mesg
// detecta que a mensagem recebida é uma mensagem Label Mapping (type=400)
OUT: Found type 400
// imprime o corpo do pacote recebido
OUT: 04 00 00 aa 00 00 00 07 01 00 00 08 02 00 01 20
OUT: c0 a8 03 01 02 00 00 04 00 00 00 10 c8 81 00 04
OUT: 83 87 ac 3d c8 80 00 86 01 03 a8 c0 00 00 ca ca
OUT: ea a7 1b 47 6b 47 ce 62 e9 53 33 83 d0 e3 14 32
OUT: 0c c4 9c e8 3c cb 2c 2f 4e f7 2b 42 3e e4 b0 ad
OUT: 31 bf f5 e2 14 1f 2c 49 6c aa 87 67 25 38 4e 69
OUT: b5 3b b2 30 ed e5 6a 48 ad ab b2 41 09 92 f2 ba
OUT: a3 42 70 3c 56 ac d2 36 ca bc f7 d0 ae f1 b8 b2
OUT: d4 49 7d fa 41 f0 27 8a 94 a4 e5 aa d6 ef 00 75
OUT: 8f 1b e6 cd 18 dd ba 36 5d 66 d5 09 83 91 09 a9
OUT: 89 17 ab cf 33 70 c7 89 dd ca ef 49 0a 3c
// tamanho do corpo da mensagem (não inclui o cabeçalho LDP)
OUT: decodedSize for Map msg = 174
// imprime o conteúdo do pacote recebido
OUT: LPD Header : protocolVersion = 1
```

```

OUT: pduLength = 180
OUT: lsrAddress = c0a80301
OUT: labelSpace = 0
OUT: LABEL MAPPING MSG ***START***:
OUT: baseMsg : uBit = 0
OUT: msgType = 400
OUT: msgLength = 170
OUT: msgId = 7
OUT: fecTlv:
OUT: Tlv:
OUT: BaseTlv: uBit = 0
OUT: fBit = 0
OUT: type = 100
OUT: length = 8
OUT: fecTlv->numberFecElements = 1
OUT: elem 0 type is 2
// FEC c0a80301/32 ou seja (192.168.3.1)
OUT: Fec Element :
OU: type = 2, addFam = 1, preLen = 32, address = c0a80301
OUT: fecTlv.wcElemExists = 0
// label genérico, diferencia de labels ATM e FrameRelay
OUT: genLblTlv:
OUT: Tlv:
OUT: BaseTlv: uBit = 0
OUT: fBit = 0
OUT: type = 200
OUT: length = 4
// label atribuído como label de saída para a FEC 192.168.3.1/32
OUT: genLbl data: label = 16
// imprime TLVs opcionais de Msgs Label Mapping que não estão presentes
OUT: Label mapping msg does not have atm label Tlv
OUT: Label mapping msg does not have fr label Tlv
OUT: Label mapping msg does not have hop count Tlv
OUT: Label mapping msg does not have path vector Tlv
OUT: Label mapping msg does not have label messageId Tlv
OUT: Label mapping msg does not have LSPID Tlv
OUT: Label mapping msg does not have traffic Tlv
// imprime o 1º TLV da autenticação fim a fim (TLV de Nonce), o qual foi
// inserido porque o LERB é o LSR de EGRESSO para a FEC 192.168.3.1/32
OUT: nonceTlv:
OUT: Tlv:
OUT: BaseTlv: uBit = 1
OUT: fBit = 1
OUT: type = 881
OUT: length = 4
// timestamp gerado pelo LERB baseado na sua hora local
OUT: nonceTlv data: Nonce = 1034717059
// imprime o 2º TLV da autenticação fim a fim (TLV de Hash), o qual foi
// inserido porque o LERB é o LSR de EGRESSO para a FEC 192.168.3.1/32
OUT: hashTlv:
OUT: Tlv:
OUT: BaseTlv: uBit = 1
OUT: fBit = 1
OUT: type = 880
OUT: length = 134
// valores dos campos do TLV de Hash. O campo hashValue está encriptado
// com a chave privada do LERB
OUT: hashTlv data:
OUT: lsrAddress = c0a80301, labelSpace = 0, hashDigest =
ÊÊÊ$ _GkGîbés3fĐã_2_Ămè<È,/N++B>ä°-1;ôâ__,Ilª+g%8Nîp;²0íâjH-<²A
' °°fBp<V-ô6Ê¼+D@ñ,²ôI}úAð'š"nãªöi
OUT: LABEL MAPPING MSG ***END***
// tamanho da msg Label Mapping decodificada + tamanho do cabeçalho LDP
OUT: Msg size: 174 (184)
// fim do processamento da mensagem
EXIT: ldp_decode_one_mesg
// le o estado da máquina
ENTER: ldp_state_machine

```

```

// referencia o estado 5, evento 5
OUT: FSM: state 5, event 5
// processa o estado corrente (5)
ENTER: ldp_state_process
// processa a mensagem LDP Label Mapping recebida
ENTER: ldp_label_mapping_process
// imprime mensagem confirmando o recebimento da mensagem Label Mapping do
// LERA em relação a FEC 192.168.3.1/32
OUT: Label Mapping Recv from c0a80301:0 for c0a80301/32
// instala na LIB o label recebido como label de saída para a FEC
// 192.168.3.1
_ldap_global_add_outlabel
OUT: Out Label Added
// inicia o processamento da autenticação fim a fim para a mensagem
// Label Mapping Recebida
ENTER: ldp_authentication_label_mapping_process
// Imprime valores usados na verificação da autenticação
PRT: lsr id = C0A80301
PRT: nonce = 1034717059, fec = C0A80301/32, lsrid = C0A80301
PRT: hash =
ÊÊÊŞ GkGÍbés3fĐã 2 Äæè<Ë,/N++B>ä°-1¿õâ__,Ilª#g%8Nip;²0íâjH-«²A
'õ°fBp<V-õ6Ê¼+D@ñ,²õI}úAð'ş"náªöi
// imprime o resultado da autenticação processada, neste caso SUCESSO
PRT: LDP Authentication Success for LSR c0a80301:0 FEC c0a80301/32
// encerra o processamento da autenticação fim a fim
EXIT: ldp_authentication_label_mapping_process
ENTER: Prepare_Label_Mapping_Attributes
EXIT: Prepare_Label_Mapping_Attributes
// encerra o processamento da mensagem Label Mapping
EXIT: ldp_label_mapping_process
// encerra o processamento do estado corrente (5)
EXIT: ldp_state_process
// encerra o processamento de verificação de estado da máquina
EXIT: ldp_state_machine
// encerra o processamento do buffer
EXIT: ldp_buf_process
...
----- Fim do Bloco de LOG -----

```

Figura 28 - Recebimento de mensagem Label Mapping pelo LERA (autenticação realizada com sucesso)

Caso a autenticação falhar, neste 3º evento o *log* de saída gerado pelo "mplsd", focando apenas o processamento da autenticação fim a fim, está abaixo listado (as linhas iniciadas por "//" são comentários em relação a saída de *log* subsequente):

```

----- Início do Bloco de LOG -----
...
// imprime mensagem confirmando o recebimento da mensagem Label Mapping do
// LERA em relação a FEC 192.168.3.1/32
OUT: Label Mapping Recv from c0a80301:0 for c0a80301/32
_ldap_global_add_outlabel
// instala na FIB o label recebido como label de saída para a FEC
// 192.168.3.1
OUT: Out Label Added
// inicia o processamento da autenticação fim a fim para a mensagem Label
// Mapping Recebida
ENTER: ldp_authentication_label_mapping_process
// Imprime valores usados na verificação da autenticação
PRT: lsr id = C0A80301
PRT: nonce = 1034717059, fec = C0A80301/32, lsrid = C0A80301
// neste exemplo a chave pública do LERA foi modificada propositalmente
// no arquivo ldp_peers do LERA (causando uma falha de autenticação), pois

```



```

// o LERA não pode decriptar o valor hashDidest corretamente, uma vez que
// a chave pública do LERB utilizada não estava correta.
// imprime o resultado da autenticação processada, neste caso FALHA
OUT: LDP Authentication failed for LSR c0a80301:0 FEC c0a80301/32
// Envia mensagem LDP Notification ao LERB, com código de status
// "LDP_AUTH_FAILED" (tipo=27), notificando a falha da autenticação
ENTER: ldp_notif_send
OUT: Notification Sent(2)
// imprime o conteúdo da mensagem LDP Notification
OUT: LPD Header : protocolVersion = 1
OUT: pduLength = 28
// LSR originador da mensagem 192.168.0.1:0 (LERA)
OUT: lsrAddress = c0a80001
OUT: labelSpace = 0
OUT: NOTIF MSG ***START***:
OUT: baseMsg : uBit = 0
OUT: msgType = 1
OUT: msgLength = 18
OUT: msgId = 8
OUT: statusTlv:
OUT: Tlv:
OUT: BaseTlv: uBit = 1
OUT: fBit = 1
OUT: type = 300
OUT: length = 10
// imprime o ID da mensagem Label Mapping que gerou esta notificação
OUT: status data: msgId = 9
OUT: msgType = 0
OUT: status Flags: error = 1
// forward deve ser 1 para ficar alinhado com o fBit do Tlv de Status
OUT: forward = 1
// código de status retornado (tipo 27 = "LDP_AUTH_FAILED")
OUT: status = 27
OUT: Notif msg does not have Extended Status TLV
OUT: Notif msg does not have Return PDU
OUT: Notif msg does not have Return MSG
OUT: NOTIF MSG ***END***:
// encerra função de envio da mensagem LDP Notification
EXIT: ldp_notif_send
// encerra o processamento da autenticação fim a fim
EXIT: ldp_authentication_label_mapping_process
...
----- Fim do Bloco de LOG -----

```

Figura 30 - Recebimento de mensagem Label Mapping pelo LERA (autenticação falhou)

1º evento) o LERA envia mensagem LDP Label Mapping para o LERB:

No 1º evento o LERA (c0a80001) envia uma mensagem LDP Label Mapping para o LERB (c0a80301) em relação a FEC 192.168.0.1/32 (em Hexadecimal c0a80001/32), que é também seu LSR-ID:

"OUT: Label Mapping Sent to c0a80301:0 for c0a80001/32".

Como o prefixo de rede IP 192.168.0.1/32 é local ao LERA (interface Lo), o LERA será o EGRESSO em relação a FEC 192.168.0.1/32. Por estar configurado no modo de distribuição NÃO SOLICITADO, onde o LSR anuncia

seus novos mapeamentos Etiqueta/FEC sem que ninguém os solicite, o LERA deve:

- gerar um label de entrada para esta FEC;
- codificar os TLVS da autenticação fim a fim (NonceTLV e HashTLV); e
- enviar uma mensagem LDP Label Mapping ao seu par LDP ativo (LERB), de forma a criar um LSP com o LERB em relação a esta FEC;

Estes passos foram executados com sucesso conforme o bloco de log gerado pela execução do "mplsd" no ambiente de testes, abaixo apresentado (as linhas iniciadas por "//" são comentários em relação a saída de log subsequente):

```

----- Inicio do Bloco de LOG -----
...
// inicia a função que lê sequencialmente a tabela de roteamento do LERA
ENTER: ldp_label_mapping_initial_callback
OUT: Initial Label Mapping fired: session(2)
ldp_label_mapping_with_xc: enter
ENTER: Prepare_Label_Mapping_Attributes
EXIT: Prepare_Label_Mapping_Attributes
// Inicia função de que Label Mapping
ENTER: ldp_label_mapping_send
// label de entrada adicionado (valor do label = 16, conforme abaixo)
OUT: In Label Added
// rotina que gera a assinatura digital (hash encriptado com a chave
// privada do LSR 192.168.0.1)
ENTER: setupHashTlv
// valores de entrada para funcao hash sha1-160 bits
PRT: setupHashTlv: nonce = 1034717280, fec = C0A80001/32, lsrid = C0A80001
// resultado da função hash
PRT: setupHashTlv: hash = "±õ6\@-!,-
// assinatura RSA, com chave de 128 bits, aplicada sobre o valor hash
PRT: setupHashTlv: signLen = 128
PRT: setupHashTlv: sign =
c_•E±_ó_ÈÖçPJoóŠ'™-¼³e¿,(™_ ;è,,ëUÇ2_ð,□Ye%éš□yï"wÉ">E_"_fâ6æèPáEÿ_&½ÿiyíÁTÔ
„SÖÉIYhÖ@üyÃ_fÁEâX,&Pm•%âü~ñi...QÉ*xâ,>†é~SâAYA_-%.□.©@p
EXIT: setupHashTlv
// imprime campos e TLVs da mensagem Label Mapping que será enviada
OUT: LDP Header : protocolVersion = 1
OUT: pduLength = 180
// LSR de origem (LERA)
OUT: lsrAddress = c0a80001
OUT: labelSpace = 0
OUT: LABEL MAPPING MSG ***START***:
OUT: baseMsg : uBit = 0
OUT: msgType = 400
OUT: msgLength = 170
OUT: msgId = 7
OUT: fecTlv:
OUT: Tlv:
OUT: BaseTlv: uBit = 0
OUT: fBit = 0
OUT: type = 100
OUT: length = 8
OUT: fecTlv->numberFecElements = 1
OUT: elem 0 type is 2
// FEC e prefixo de rede (192.168.0.1/32)

```

```

OUT:          Fec Element : type = 2, addFam = 1, preLen = 32, address =
c0a80001
OUT:
OUT:          fecTlv.wcElemExists = 0
// tipo do label (ethernet) outras opção possíveis são ATM e Frame Relay
OUT:          genLblTlv:
OUT:          Tlv:
OUT:          BaseTlv: uBit = 0
OUT:                  fBit = 0
OUT:                  type = 200
OUT:                  length = 4
// label atribuído como label de entrada para a FEC 192.168.0.1/32
OUT:          genLbl data: label = 16
// imprime TLVs opcionais da mensagem Label Mapping
OUT:          Label mapping msg does not have atm label Tlv
OUT:          Label mapping msg does not have fr label Tlv
OUT:          Label mapping msg does not have hop count Tlv
OUT:          Label mapping msg does not have path vector Tlv
OUT:          Label mapping msg does not have label messageId Tlv
OUT:          Label mapping msg does not have LSPID Tlv
OUT:          Label mapping msg does not have traffic Tlv
// imprime o 1º TLV da autenticação fim a fim (TLV de Nonce), o qual foi
// inserido porque o LERA é o LSR de EGRESSO para a FEC 192.168.0.1/32
OUT:          nonceTlv:
OUT:          Tlv:
OUT:          BaseTlv: uBit = 1
OUT:                  fBit = 1
OUT:                  type = 881
OUT:                  length = 4
OUT:          nonceTlv data: Nonce = 1034717280
// timestamp gerado pelo LERA baseado na hora local
OUT:          nonceTlv data: Nonce = 1034717280
// imprime o 2º TLV da autenticação fim a fim (TLV de Hash), o qual foi
// inserido porque o LERA é o LSR de EGRESSO para a FEC 192.168.0.1/32
OUT:          hashTlv:
OUT:          Tlv:
OUT:          BaseTlv: uBit = 1
OUT:                  fBit = 1
OUT:                  type = 880
OUT:                  length = 134
// Campos do TLV de Hash. (hashValue está encriptado com a chave
// privada do LERA)
OUT:          hashTlv data:
OUT:          lsrAddress = C0A80001, labelSpace = 0, hashDigest =
c_•E+_ó_ÈÒçPJoöŠ'™-¾³eç„(™_;è„èUÇ2_ð_□Ye%és□yi"WE">E_"_ "fâ6æèPáEÿ_&¼ÿyiÁTÔ
„SÖEÿhÖ@ùyÃ_fÃÈäX,&Pm•%äü-ñi...Gé*xâ,>†é~SâÄYA_=%□.©p
OUT: LABEL MAPPING MSG ***END***:
// imprime mensagem confirmando envio da mensagem Label Mapping para o
// LERB em relação a FEC 192.168.0.1/32
OUT: Label Mapping Sent to c0a80301:0 for c0a80001/32
// encerra a função de envio de Label Mapping
EXIT: ldp_label_mapping_send
      ldp_label_mapping_with_xc: exit
// encerra a função que leu sequencialmente a tabela de roteamento do LERA
EXIT: ldp_label_mapping_initial_callback
...
----- Fim do Bloco de LOG -----

```

Figura 31 - Envio de mensagem Label Mapping do LERA para o LERB

4º evento) o LERB recebe uma mensagem LDP Label Mapping do LERA e a autenticação é processada pela segunda e última vez no ambiente:

No 4º evento o LERB recebe uma mensagem LDP Label Mapping do LERA em relação a FEC 192.168.0.1/32 (em Hexadecimal c0a80001/32):

"OUT: Label Mapping Recv from c0a80001:0 for c0a80001/32".

Como existe uma entrada na tabela de roteamento do LERB indicando que o LERA é o próximo hop para a FEC 192.168.0.1/32, o LERB detecta que é o INGRESSO para esta FEC, dessa forma deve:

- adicionar o label recebido como label de saída para a FEC 192.168.0.1/32;
- processar os TLVS da autenticação fim a fim (NonceTLV e HashTLV) de modo a validar se o LERA tem autorização para estabelecer LSPs com o LERB; e
- estabelecer o LSP com o LERA, dessa forma o LSP terá o como INGRESSO, o LERB, e o como EGRESSO , o LERA. O sentido do LSP será do LERB em direção ao LERA. Dessa forma criando um LSP inverso ao criado no 3º evento, permitindo uma comunicação bidirecional entre o LERA e o LERB.

Estes passos foram executados com sucesso conforme o bloco de log gerado pela execução do "mplsd" no ambiente de testes, abaixo apresentado (as linhas iniciadas por "//" são comentários em relação a saída de log subsequente):

```
----- Início do Bloco de LOG -----
...
// novo evento detectado
ENTER: ldp_event
// inicia processamento do buffer de entrada
ENTER: ldp_buf_process
// inicia decodificação do cabeçalho LDP e mostra o resultado
ENTER: ldp_decode_header
OUT: 00 01 00 b4 c0 a8 00 01 00 00
// encerra o processamento do cabeçalho LDP
EXIT: ldp_decode_header
// inicia decodificação do corpo da mensagem recebida
ENTER: ldp_decode_one_msg
// detecta que a mensagem recebida é uma mensagem LDP Label Mapping
OUT: Found type 400
// imprime o corpo do pacote recebido
OUT: 04 00 00 aa 00 00 00 07 01 00 00 08 02 00 01 20
OUT: c0 a8 00 01 02 00 00 04 00 00 00 10 c8 81 00 04
OUT: 60 88 ac 3d c8 80 00 86 01 00 a8 c0 00 00 63 1a
OUT: 95 45 b1 1a f3 11 c8 d2 a2 de 4a 6f f6 8a 27 99
OUT: ad be b3 65 bf 84 28 99 0c 3b e8 84 eb 55 c7 32
OUT: 16 f0 b8 90 a5 65 25 e9 9a 9e 79 cf 94 77 c9 93
OUT: 3e 45 15 94 0e 22 a3 e5 36 e6 e8 50 e1 45 ff 01
OUT: 26 bd cc 79 ed c1 54 d4 84 53 d6 c9 ee a5 68 d6
OUT: 40 f9 79 c3 14 83 c3 c8 e4 58 2c 26 50 6d 7f bc
OUT: e4 fc ac f1 ef 85 8c e9 2a d7 e5 2c 3e 86 e9 7e
OUT: 53 e5 c0 59 41 11 3d 89 b6 90 2e 40 a9 70
// tamanho do corpo da mensagem (não inclui o cabeçalho LDP)
OUT: decodedSize for Map msg = 174
// imprime o conteúdo do pacote recebido
OUT: LDP Header : protocolVersion = 1
OUT: pduLength = 180
OUT: lsrAddress = c0a80001
```

```

OUT:      labelSpace = 0
OUT: LABEL MAPPING MSG ***START***:
OUT:      baseMsg : uBit = 0
OUT:              msgType = 400
OUT:              msgLength = 170
OUT:              msgId = 7
OUT:      fecTlv:
OUT:      Tlv:
OUT:      BaseTlv: uBit = 0
OUT:              fBit = 0
OUT:              type = 100
OUT:              length = 8
OUT:              fecTlv->numberFecElements = 1
OUT:              elem 0 type is 2
// FEC c0a80001/32 ou seja (192.168.0.1)
OUT:      Fec Element :
OUT:              type = 2, addFam = 1, preLen = 32, address = c0a80001
OUT:      fecTlv.wcElemExists = 0
OUT:      genLblTlv:
OUT:      Tlv:
OUT:      BaseTlv: uBit = 0
OUT:              fBit = 0
OUT:              type = 200
OUT:              length = 4
// valor do label atribuído como label de saída para a FEC 192.168.0.1/32
OUT:      genLbl data: label = 16
// imprime TLVs opcionais de Msgs Label Mapping que não estão presentes
OUT:      Label mapping msg does not have atm label Tlv
OUT:      Label mapping msg does not have fr label Tlv
OUT:      Label mapping msg does not have hop count Tlv
OUT:      Label mapping msg does not have path vector Tlv
OUT:      Label mapping msg does not have label messageId Tlv
OUT:      Label mapping msg does not have LSPID Tlv
OUT:      Label mapping msg does not have traffic Tlv
// imprime o 1º TLV da autenticação fim a fim (TLV de Nonce), o qual foi
// inserido porque o LERA é o LSR de EGRESSO para a FEC 192.168.0.1/32
OUT:      nonceTlv:
OUT:      Tlv:
OUT:      BaseTlv: uBit = 1
OUT:              fBit = 1
OUT:              type = 881
OUT:              length = 4
OUT:      nonceTlv data: Nonce = 1034717280
// timestamp gerado pelo LERA baseado na sua hora local
OUT:      nonceTlv data: Nonce = 1034717280
// imprime o 2º TLV da autenticação fim a fim (TLV de Hash), o qual foi
// inserido porque o LERA é o LSR de EGRESSO para a FEC 192.168.0.1/32
OUT:      hashTlv:
OUT:      Tlv:
OUT:      BaseTlv: uBit = 1
OUT:              fBit = 1
OUT:              type = 880
OUT:              length = 134
// valores dos campos do TLV de Hash. O campo hashValue está encriptado
// com a chave privada do LERA
OUT:      hashTlv data:
OUT:      lsrAddress = C0A80001, labelSpace = 0, hashDigest =
c_•E±_ó_ÈÒçPJoöŠ'™-¾³eç,,(™_ ;è,,ëUÇ2_ð,□Ye%és□yi"WE">E_"_ "fâ6æèPáEÿ_&½ÿiyiÁTÔ
„SÖÊÿhÖ@ùyÃ_ fÃÊäX,&Pm•%äü-ñi...Gé*×â,>†é~SâÀYA_=%□□.©@p
OUT: LABEL MAPPING MSG ***END***:
// tamanho da msg Label Mapping decodificada + cabeçalho LDP
OUT: Msg size: 174 (184)
// fim do processamento da mensagem
EXIT: ldp_decode_one_mesg
// le o estado da máquina
ENTER: ldp_state_machine
// processa o estado corrente (5)
OUT: FSM: state 5, event 5

```

```

ENTER: ldp_state_process
// processa a mensagem LDP Label Mapping recebida
ENTER: ldp_label_mapping_process
// imprime mensagem confirmando o recebimento da mensagem Label Mapping
// enviada pelo LERA em relação a FEC 192.168.3.1/32
OUT: Label Mapping Recv from c0a80001:0 for c0a80001/32
// instala na FIB o label recebido como label de saída para a FEC
// 192.168.0.1
_ldap_global_add_outlabel
OUT: Out Label Added
// inicia o processamento da autenticação fim a fim para a mensagem Label
// Mapping recebida
ENTER: ldp_authentication_label_mapping_process
// Imprime valores usados na verificação da autenticação
PRT: lsr id = C0A80001
PRT: nonce = 1034717280, fec = C0A80001/32, lsrid = C0A80001
PRT: hash =
c_•E±_ō_ÈÖçPJoöŠ'™-¼³e¿„(™_ ;è„ëUÇ2_ð_□Ye%éš□yİ"wÉ">E_"_ "£â6æèPáEÿ_&¼ÿyíÁTÔ
„SÖÉiYhÖ@üyÃ_fÃÈäX,&Pm•¼äü-ñi...Çé*xâ,>té~SâÀYA_-%.□.□@p
// imprime o resultado da autenticação processada, neste caso SUCESSO
PRT: LDP Authentication Success for LSR c0a80001:0 FEC c0a80001/32
// encerra o processamento da autenticação fim a fim
EXIT: ldp_authentication_label_mapping_process
ENTER: Prepare_Label_Mapping_Attributes
EXIT: Prepare_Label_Mapping_Attributes
// encerra o processamento da mensagem Label Mapping
EXIT: ldp_label_mapping_process
// encerra o processamento do estado de máquina corrente (5)
EXIT: ldp_state_process
// encerra o processamento de verificação de estado da máquina
EXIT: ldp_state_machine
// encerra o processamento do buffer
EXIT: ldp_buf_process
...
----- Fim do Bloco de LOG -----

```

Figura 32 - Recebimento de mensagem Label Mapping pelo LERB (autenticação realizada com sucesso)

Caso a autenticação falhar, neste 4º evento o log de saída gerado pelo "mplsd", focando apenas o processamento da autenticação fim a fim, está abaixo listado (as linhas iniciadas por "/" são comentários em relação a saída de log subsequente):

```

----- Inicio do Bloco de LOG -----
...
// imprime mensagem confirmando o recebimento da mensagem Label Mapping do
// LERA em relação a FEC 192.168.0.1/32
OUT: Label Mapping Recv from c0a80001:0 for c0a80001/32
_ldap_global_add_outlabel
// instala na FIB o label recebido como label de saída para a FEC
192.168.0.1
OUT: Out Label Added
// inicia o processamento da autenticação fim a fim para a mensagem Label
// Mapping Recebida
ENTER: ldp_authentication_label_mapping_process
// Imprime valores usados na verificação da autenticação
PRT: lsr id = C0A80001
PRT: nonce = 1034717280, fec = C0A80001/32, lsrid = C0A80001
// neste exemplo a chave pública do LERA foi modificada propositalmente
// no arquivo ldp_peers do LERB (causando uma falha de autenticação), pois
// o LERB não pode decriptar o valor hashDidest corretamente, uma vez que
// a chave pública do LERA utilizada não estava correta

```

```

// imprime o resultado da autenticação processada, neste caso FALHA
OUT: LDP Authentication failed for LSR c0a80001:0 FEC c0a80001/32
// Envia mensagem LDP Notification ao LERA, com código de status
// "LDP_AUTH_FAILED" (tipo=27), notificando a falha da autenticação
ENTER: ldp_notif_send
OUT: Notification Sent(2)
// imprime o conteúdo da mensagem LDP Notification
OUT: LDP Header : protocolVersion = 1
OUT:   pduLength = 28
// LSR originador da mensagem 192.168.3.1:0 (LERB)
OUT:   lsrAddress = c0a80301
OUT:   labelSpace = 0
OUT: NOTIF MSG ***START***:
OUT:   baseMsg : uBit = 0
OUT:           msgType = 1
OUT:           msgLength = 18
OUT:           msgId = 8
OUT:   statusTlv:
OUT:   Tlv:
OUT:   BaseTlv: uBit = 1
OUT:           fBit = 1
OUT:           type = 300
OUT:           length = 10
// imprime o ID da mensagem Label Mapping que gerou esta notificação
OUT:   status data:   msgId = 7
OUT:                 msgType = 0
OUT:   status Flags:  error = 1
// forward deve ser 1 para ficar alinhado com o fBit do Tlv de Status
OUT:   forward = 1
// código de status retornado (tipo 27 = "LDP_AUTH_FAILED")
OUT:   status = 27
OUT:   Notif msg does not have Extended Status TLV
OUT:   Notif msg does not have Return PDU
OUT:   Notif msg does not have Return MSG
OUT: NOTIF MSG ***END***:
// encerra função de envio da mensagem LDP Notification
EXIT: ldp_notif_send
// encerra o processamento da autenticação fim a fim
EXIT: ldp_authentication_label_mapping_process
...
----- Fim do Bloco de LOG -----

```

Figura 33 - Recebimento de mensagem Label Mapping pelo LERB (autenticação falhou)

Através da análise dos resultados obtidos no ambiente de testes, pôde-se constatar que a autenticação proposta para o protocolo LDP neste trabalho é perfeitamente viável e possui aspectos que vem a acrescentar as atuais funcionalidades deste protocolo, especialmente em relação a segurança do ambiente.

4 – CONCLUSÕES

Concluiu-se que a solução de autenticação fim a fim para o protocolo LDP apresentada neste trabalho, trouxe incrementos importantes que vem a suprir lacunas e deficiências dentro da atual especificação deste protocolo, especialmente com relação à segurança do ambiente. Estas modificações no protocolo influem diretamente na melhoria da segurança de um domínio MPLS, pois prevê autenticação, integridade e proteção contra ataques de repetição.

Participar do desenvolvimento do protótipo incrementou de forma significativa os conhecimentos do autor sobre a tecnologia de rede MPLS e do protocolo LDP. Além disso, foi possível conhecer softwares e ferramentas que se tinha pouco ou nenhum conhecimento.

A atividade desenvolvida no Trabalho de Conclusão de Curso cumpriu com seu objetivo de envolver o aluno em um ambiente científico, participar de um grupo de trabalho, fazer pesquisas sobre um área da computação e desenvolver soluções.

5 – REFERÊNCIAS BIBLIOGRÁFICAS

[Abadi, 1996] ABADI, M.; NEEDHAM, R. Prudent Engineering. Practice for Cryptographic Protocols. IEEE Transactions on Software Engineering, v. 22, n. 1, p. 6-15, 1996. Disponível por <http://www.cs.virginia.edu/~survive/DOCS/prudent.ps>. Acesso em 10 set. 2002.

[Ayame, 1999] Ayame Project – MPLS Implementation for NetBSD. Início das atividades em outubro de 1999. Disponível por <http://www.ayame.org/AYAMEproject.php>. Acesso em 9 Mai. 2002.

[Andersson, 2001] ANDERSSON, L.; Doolan, P., Feldman, N., et al. LDP Specification. RFC 3036, janeiro de 2001. Disponível por <http://www.ietf.org/rfc/rfc3036.txt>. Acesso em 20 nov. 2001.

[Awduche, 2001] AWDUCHE, D; BERGER, L RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209, dezembro de 2001. Disponível por <http://www.ietf.org/rfc/rfc3209.txt>. Acesso em 25 mar. 2002.

[Badan, 2001] BADAN, Tomás. A. C.; PRADO, Rodrigo C. M.; ZAGARI, Eduardo N. F.; et. al. Uma Implementação MPLS para Redes Linux. Universidade de Campinas (UNICAMP), DCA – FEEC. Publicado no SBRC 2001, maio de 2001. Disponível por <http://www.sbrc2001.ufsc.br/artigos/4807-7171.pdf>. Acesso em 15 abr. 2002.

[Buda, 2001] BUDA, G.; CHOI, D.; et. al. Security Standarts for the Global Information Grid. IFIP/IEEE International Symposium on Integrated Network Management, Seattle, Maio de 2001.

[Comer, 1998] COMER D. Interligação em rede com TCP/IP. Volume I, 3a Ed. - Rio de Janeiro: Campus, 1998.

[Comer, 1999] COMER D. Interligação em rede com TCP/IP. Volume II, 3a Ed. - Rio de Janeiro: Campus, 1999.

[De Clercq, 2001] DE CLERCQ, J.; PARIDAENS, O.; T'JOENSET Y., SCHRIJVER, P. End to End Authentication for LDP. Draft-schrijvp-mpls-ldp-end-to-end-auth-03.txt. jeremy.de_clercq@alcatel.be, fevereiro de 2001. Contato com o author em 10 jan. 2002. A Draft expirou, cf.. <http://www.ietf.org/internet-drafts/draft-schrijvp-mpls-ldp-end-to-end-auth-04.txt>.

[Dobbertin, 1996] DOBBERTIN, H. The Status of MD5 After a Recent Attack. CryptoBytes Vol. 2, No.2, edição de verão, 1996. Disponível por <ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf>. Acesso em 19 dez 2001.

[Heffernan, 1998] HEFFERNAN, A. Protection of BGP Sessions via the TCP MD5 Signature Option. RFC 2385, agosto de 1998. Disponível por <http://www.ietf.org/rfc/rfc2385.txt>. Acesso em 25 jan 2002.

- [Jamoussi, 2002] JAMOUISSI, B, et al. Constraint-Based LSP Setup using LDP. RFC 3212, janeiro de 2002. Disponível por <http://www.ietf.org/rfc/rfc3212.txt>. Acesso em 12 fev. 2002.
- [Kent, 2000] KENT, S.; LYNN, C.; SEO, K. Secure Border Gateway Protocol (S-BGP). IEEE Journal on Selected Areas In Communications, VOL. 18, NO. 4, abril de 2000.
- [Leu, 2000] LEU, J; et. al. Project: MPLS for Linux. Início das atividades em 27 de novembro de 2000. Disponível por <http://sourceforge.net/projects/mps-linux>. Acesso em 06 fev. 2002. (Trabalho em progresso).
- [Magalhães, 2001] MAGALHÃES, M. F.; CARDOZO, E. Introdução a Comutação ao IP por Rótulos Através de MPLS. Universidade de Campinas (UNICAMP), DCA – FEEC. Publicado no SBRC 2001, Minicursos, maio de 2001.
- [Müller, 2002] MÜLLER, M. Uma Solução de Autenticação Fim a Fim para o LDP (Label Distribution Protocol). Florianópolis. Novembro de 2002.
- [Nist, 2000] NIST - National Institute for Standards and Technology. Descriptions of SHA-256, SHA-384, and SHA-512. Outubro de 2000. Disponível por <http://csrc.nist.gov/cryptval/shs/sha256-384-512.pdf>. Acesso em 20 mai. 2002.
- [Rekhter, 2001] REKHTER, Y; ROSEN, E. Carrying Label Information in BGP-4. RFC 3107, maio de 2001. Disponível por <http://www.ietf.org/rfc/rfc3107.txt>. Acesso em 22 jan. 2002.
- [Rivest, 1992] RIVEST, R. The MD5 Message-Digest Algorithm. RFC 1321, abril de 1992. Disponível por <http://www.ietf.org/rfc/rfc1321.txt>. Acesso em 25 jan. 2002.
- [Rodriguez, 2001] Rodriguez, A.; Gattrell J.; Karas J.; Peschke R. TCP/IP Tutorial and Technical Overview. 2001. Disponível por <http://www.ibm.com/redbooks>. Acesso em 08 de março de 2003.
- [Rosen, 2002] ROSEN, E.; TAPPAN, D.; FEDORKOW, G., et al. MPLS Label Stack Encoding. RFC 3032, janeiro de 2002a. Disponível por <http://www.ietf.org/rfc/rfc3032.txt>. Acesso em 12 set. 2001.
- [Rosen, 2001] ROSEN, E; VISWANATHAN, A.; CALLON, R. Multiprotocol Label Switching Architecture. RFC 3031, janeiro de 2001. Disponível por <http://www.ietf.org/rfc/rfc3031.txt>. Acesso em 26 jul. 2001
- [Stallings, 1999] STALLINGS, William. Cryptography and Network Security: Principles and Practice. New Jersey, 1999, editora Prentice-Hall, Inc., 2ª ed.
- [Tanenbaum, 1995] TANENBAUM, A. Sistemas Operacionais Modernos. São Paulo: Ed. Campus, 1995.