UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Morvan Daniel Müller

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

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Carlos Becker Westphall
Orientador

Carla Merkle Westphall
Co-orientadora

Florianópolis, Novembro de 2002.

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

Morvan Daniel Müller

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Fernando A. O. Gauthier, Dr.
Coordenador

Banca Examinadora

_____

Carlos Becker Westphall, Dr.
Orientador

_____

Carla Merkle Westphall, Dr.
Co-Orientadora

_____

Ricardo Felipe Custódio, Dr.
Membro da Banca

_____

Joni da Silva Fraga, Dr.
Membro da Banca

AGRADECIMENTOS

Em especial gostaria de agradecer, pelas muitas contribuições prestadas, aos professores deste curso de mestrado Prof. Dr. Carlos Becker Westphall, Profa. Dra. Carla Merkle Westphall e ao Prof. Dr. Ricardo Felipe Custódio.

Gostaria também de prestar o meu reconhecimento pela grande colaboração prestada, por e-mail, pelo Sr. Jeremy De Clercq, do grupo de Estratégias de Redes da empresa Alcatel da Bélgica, a sua colaboração foi fundamental para o meu entendimento de conceitos técnicos avançados do LDP e pelo vislumbramento da problemática para a qual sugeri a solução escopo deste trabalho.

Por final gostaria de agradecer pelas colaborações prestadas na fase de implementação ao Sr. James Leu, desenvolvedor do projeto Mpls-Linux da source forge e aos graduandos em Ciência da Computação da UFSC, Leonardo Neves Bernardo e Bruno Bisol.

SUMÁRIO

LISTA DE ABREVIATURAS ..... **VII**
LISTA DE FIGURAS ..... **VIII**
LISTA DE TABELAS ..... **X**
RESUMO ..... **XI**
1 INTRODUÇÃO **1**
1.1 Motivação 1
1.2 Objetivos 2
1.2.1 Objetivo Geral 2
1.2.2 Objetivos Específicos 2
1.3 Metodologia 3
1.4 Organização do Trabalho 3
2 MPLS (MULTI-PROTOCOL LABEL SWITCHING) **4**
2.1 Introdução 4
2.2 Características e Funcionamento do MPLS 5
2.3 Conceitos Básicos no MPLS 8
2.3.1 FEC (*Forward Equivalence Class*) 8
2.3.2 LER (*Label Edge Router*) 9
2.3.3 LSR (*Label Switching Router*) 9
2.3.4 LSP (*Label Switch Path*) 10
2.4 Roteamento de LSPs 10
2.4.1 Roteamento *hop-by-hop* 10
2.4.2 Roteamento Explícito 11
2.5 Distribuição de Etiquetas no MPLS 12
2.5.1 Etiquetas MPLS 12
2.5.2 Pilha de Etiquetas 13
2.5.3 Pacote Etiquetado 15
2.5.4 Protocolos de Distribuição de Etiquetas 15
2.5.5 Estratégias de Distribuição de Etiquetas 16
2.6 Conclusões do Capítulo 20
3 LDP (*LABEL DISTRIBUTION PROTOCOL*) **21**
3.1 Características e Funcionamento do LDP 21
3.2 PDUs (*Protocol Data Units*) LDP e TLVs (*Type-Length-Value)* 22
3.3 Categorias de Mensagens LDP 24
3.4 Estabelecimento de Sessões LDP 25
3.5 Manutenção e Encerramento das Sessões LDP 27
3.6 Gerência de Etiquetas 28
3.7 Planos de Controle e Encaminhamento no LDP 30
3.8 CR-LDP (*Constrained-Based Routing Protocol*) 31
3.9 Conclusões do Capítulo 32
4 CONSIDERAÇÕES SOBRE A SEGURANÇA DO LDP **33**

4.1 Ataques Baseados em *Spoofing* ... 33
4.1.1 Troca de Mensagens de Descoberta Transportadas via UDP ... 33
4.1.2 Sessão LDP Transportada via TCP ... 34
4.2 Confidencialidade ... 34
4.3 Negação de Serviço (*Denial of Service*) ... 34
4.3.1 Porta UDP 646 usada pelo Mecanismo de Descoberta do LDP ... 35
4.3.2 Estabelecimento de Sessões LDP na Porta TCP 646 ... 35
4.4 Considerações sobre Autenticação e Integridade no LDP ... 36
4.5 Requisitos de Segurança para o LDP ... 36
4.6 Trabalhos Correlatos ... 37
4.7 Conclusões do Capítulo ... 44
5 AUTENTICAÇÃO FIM A FIM PARA O LDP ... 45
5.1 Estabelecimento de um LSP (*Label Switching Path*) entre LSRs (*Label Switching Routers*) Não-adjacentes ... 45
5.2 Proposta de Autenticação Fim a Fim para o LDP ... 49
5.2.1 Introdução ... 49
5.2.2 Modelo de Autenticação Proposto ... 51
5.2.3 Novos TLVs e Tipos Definidos ao LDP pela Solução de Autenticação ... 53
5.2.3.1 TLV de Hash ... 53
5.2.3.2 TLV de Nonce ... 54
5.2.3.3 Novo Código de Status "*Authentication Failed*" ... 55
5.2.3.4 Considerações sobre o Registro dos TLVs Definidos pela Solução ... 56
5.2.4 Procedimentos da Autenticação Fim a Fim ... 56
5.2.4.1 Procedimentos ao ENVIAR Mensagens LDP ... 56
5.2.4.2 Procedimentos ao RECEBER Mensagens LDP ... 58
5.3 Discussão sobre a Solução de Autenticação Proposta ... 59
5.4 Considerações sobre Algoritmos de função Hash e Criptografia Assimétrica Sugeridos para a Solução ... 62
5.4.1 Algoritmos de *Hash* ... 62
5.4.2 Algoritmos de Criptografia Assimétrica ... 62
5.5 Solução para Distribuição de Chaves usando Certificação Digital ... 63
5.6 Conclusões do Capítulo ... 66
6 RESULTADOS DA IMPLEMENTAÇÃO ... 67
6.1 Ferramentas Utilizadas ... 67
6.2 Definições Adotadas na Implementação do Protótipo ... 68
6.3 Descrição da Implementação da Solução de Autenticação ... 70
6.4 Instalação das Ferramentas no Linux ... 90
6.5 Configuração e Execução do LDP no Ambiente de Testes ... 92
6.5.1 Visualização do Funcionamento do LDP no Ambiente de Testes ... 93
6.6 Resultados Obtidos ... 95
6.7 Conclusões do Capítulo ... 111
7 CONCLUSÕES ... 112
7.1 Revisão das Motivações e Objetivos ... 112
7.2 Visão Geral do Trabalho ... 112
7.3 Contribuições e Escopo do Trabalho ... 113
7.4 Perspectivas Futuras ... 114
8 REFERÊNCIAS BIBLIOGRÁFICAS ... 115

ANEXO 1..... 118
ANEXO 2..... 120
ANEXO 3..... 125

LISTA DE ABREVIATURAS

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

LISTA DE FIGURAS

Figura 1. Conceitos básicos no MPLS (FEC, LER, LSR e LSP). 8

Figura 2. Uma Etiqueta MPLS posicionada dentro de um pacote de enlace. 13

Figura 3. Exemplo de uma pilha de etiquetas em um pacote da rede. Observe que o campo “Stack Bit” do último pacote possui o valor 1. 13

Figura 4. Pilhas de etiquetas: Túneis e Hierarquias [MAGALHÃES, 2001] 15

Figura 5. Distribuição Não Solicitada..... 18

Figura 6. Distribuição Sob Demanda..... 18

Figura 7. Formato do Cabeçalho LDP de uma PDU LDP..... 22

Figura 8. Formato de um TLV de uma PDU LDP..... 23

Figura 9. Exemplos de troca de mensagens LDP [MAGALHÃES, 2001]..... 25

Figura 10. Manutenção e encerramento de uma sessão LDP 27

Figura 11. Passos executados pelo LDP para estabelecer um LSP entre os LSRs não-adjacentes LERA e LERB..... 46

Figura 12. Visão do plano de controle do LDP após o estabelecimento das sessões TCP/LDP (fase ativa do LDP) 47

Figura 13. Visão do Plano de Encaminhamento do LDP após o estabelecimento do LSP entre LERA e o LERB 48

Figura 14. Diagrama da autenticação fim a fim proposta para o LDP. 51

Figura 15. TLV de Hash 53

Figura 16. TLV de Nonce 55

Figura 17. Entradas para mensagens LDP LABEL REQUEST e LABEL MAPPING..... 57

Figura 18. Entrada para mensagens LDP NOTIFICATION 57

Figura 19. Exemplo de TLV de Certificado Digital 65

Figura 20. Ambiente de testes utilizado para verificar a autenticação implementada. 92

Figura 21. Diagrama de seqüência das mensagens de troca de *labels* ocorridas durante a execução do ambiente de testes. 98

Figura 22. Envio de mensagem Label Mapping do LERB para o LERA 101

Figura 23. Recebimento de mensagem Label Mapping pelo LERA (autenticação realizada com sucesso)..... 104

Figura 24. Recebimento de mensagem Label Mapping pelo LERA (autenticação falhou) 105

Figura 25. Envio de mensagem Label Mapping do LERA para o LERB..... 107

Figura 26. Recebimento de mensagem Label Mapping pelo LERB (autenticação realizada com sucesso) 110
Figura 27. Recebimento de mensagem Label Mapping pelo LERB (autenticação falhou) 111

LISTA DE TABELAS

Tabela 1. Resumo das Estratégias de Distribuição de Etiquetas 20

RESUMO

Este trabalho propõe uma solução de autenticação para o protocolo LDP (Label Distribution Protocol) que tem por objetivo autenticar, em um escopo fim a fim, o estabelecimento de um LSP (Label Switching Path) entre um LSR (Label Switching Router) de Ingresso e o seu respectivo LSR de Egresso. Objetiva-se suprir a deficiência do protocolo LDP de não possuir um mecanismo de autenticação fim a fim definido, aplicável entre LSRs não-adjacentes.

Conforme foi verificado pelo levantamento de trabalhos correlatos, atualmente é desconhecida uma solução de autenticação semelhante, que efetivamente atenda o propósito de autenticar num escopo fim a fim, o estabelecimento de LSPs no protocolo LDP. Dessa forma a solução deste trabalho é inédita no seu escopo de aplicação.

A solução foi planejada para ambientes onde LSPs atravessam múltiplos domínios externos, não confiáveis entre si, e que por isso necessitam de um mecanismo de autenticação durante o estabelecimento dos LSPs.

A solução faz uso de um mecanismo de autenticação, baseado em criptografia assimétrica (chave pública e privada), anexado a cada mensagem LDP. Este mecanismo possibilita ao LSR receptor verificar e autenticar o originador da mensagem LDP. Adicionalmente a solução provê integridade de dados através de um mecanismo de resumo de mensagens (hash) e também protege contra ataques de repetição através da inserção de um nonce às mensagens LDP.

A solução foi projetada para ser utilizada com IP versão 4 (ipv4), requer que o protocolo LDP opere no Modo de Controle Ordenado e pode ser adicionalmente aplicada ao protocolo CR-LDP (Constraint-based Routed Label Distribution Protocol).

A solução de autenticação proposta por este trabalho, foi implementada em um ambiente MPLS de código aberto, na plataforma Linux.

Palavras-chave: *LDP, MPLS, autenticação, integridade, segurança.*

ABSTRACT

This works propose a solution for the LDP (Label Distribution Protocol) protocol to authenticate, on an end to end basis, the establishment of an LSP (Label Switching Path) between the Ingress LSR and its Egress. The objective is to supply the LDP protocol deficiency, that doesn't have one end to end authentication mechanism defined for non-adjacent LSRs.

As verified by the related work study, it is unknowed one similar authentication solution, that really address the purpose to authenticate end to end, the establishment of LSPs in the LDP protocol. So, the solution presented in this work is unpublished into its application scope.

The solution was planned for environments where the LSPs go across mutiple external domains, not trustworthy between them, and because of this need one authentication mechanism during the LSPs establishment.

The solution makes use of one digital signature mechanism from the data source, based on asymmetric cryptography, attached to each LDP message. This mechanism enable the receiver to verify and authenticate the sender of the message. Additionally the solution provides data integrity by the use of a hash and protects against replay attacks inserting a nonce to the LDP messages.

The solution was planned to be used with IPv4 (IP version 4), require that the LDP operates on the Ordered Control Mode and can additionally be applied to the CR-LDP (Constraint-based Routed Label Distribution) Protocol.

The solution proposed in this work was implemented in a open source MPLS environment for Linux.

***Key-words**: LDP, MPLS, authentication, integrity, security.*

1 INTRODUÇÃO

1.1 Motivação

O MPLS *(Multiprotocol Label Switchin*g), conforme a RFC3031 [ROSEN, 2001], é um *framework* definido pelo IETF *(Internet Engineering Task Force*) que proporciona designação, encaminhamento e comutação eficientes de fluxos de tráfego através da rede, por ser uma técnica de comutação de pacotes baseada em etiquetas (*labels*). Na arquitetura MPLS, o protocolo LDP (*Label Distribution Protocol*) é responsável pela distribuição das etiquetas e pelo estabelecimento de caminhos lógicos chamados LSPs (*Label Switched Paths*). Estes caminhos lógicos são criados por roteadores LSRs (*Label Switched Routers*) interligados entre si.

Como o LDP exerce uma função primordial dentro do ambiente MPLS, uma lacuna na segurança do LDP pode comprometer todo o ambiente, tendo em vista que a distribuição das etiquetas (*labels*), realizada pelo LDP, é o que determina quem pode participar ou não do domínio MPLS.

A forma de autenticação definida para o LDP [ANDERSSON, 2001], 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.

Desta forma, este trabalho propõe uma solução de autenticação fim a fim para o protocolo LDP, de modo a preencher esta lacuna do protocolo, viabilizando o estabelecimento de LSPs entre dois LSRs não-adjacentes.

A solução de autenticação fim a fim, foi planejada para ambientes onde LSPs atravessam múltiplos domínios externos, considerados não confiáveis, e que por isso necessitam se autenticar durante o estabelecimento de um novo LSP.

Conforme foi verificado pelo levantamento de trabalhos correlatos [DE CLERCQ, 2001], [BUDA, 2001], [KENT, 2000], atualmente é desconhecida uma solução de

autenticação semelhante, que efetivamente atenda o propósito de autenticar num escopo fim a fim, o estabelecimento de LSPs dentro do protocolo LDP. Dessa forma a solução proposta por este trabalho é inédita dentro do seu escopo de aplicação.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é analisar a situação atual dos mecanismos de segurança do MPLS, visando contribuir para o aperfeiçoamento destes mecanismos.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

definir mecanismos no protocolo LDP que viabilizem a autenticação mútua entre dois LSRs não-adjacentes, num escopo fim a fim, durante o estabelecimento de um novo LSP;

definir um mecanismo de autenticação apropriado para suprir as necessidades do ambiente LDP.

Não se tem o objetivo de substituir a atual opção de autenticação, baseada em na opção TCP/MD5, definida para o LDP na RFC 3036 [ANDERSSON, 2001], porque os escopos de aplicação das duas soluções são diferenciados, embora possam servir ao mesmo propósito em certas situações. A solução de autenticação baseada em TCP/MD5 só pode ser aplicada entre dois LSRs adjacentes pois depende de uma conexão TCP entre os LSRs envolvidos. No caso de LSRs não-adjacentes, durante o estabelecimento do primeiro LSP, não existe uma conexão TCP fim a fim entre os LSRs, desta forma a solução da RFC 3036 não se aplica a este caso.

1.3 Metodologia

O desenvolvimento do trabalho inclui três fases: revisão bibliográfica, planejamento e especificação da solução de autenticação, e validação da proposta através da implementação de um protótipo em um ambiente MPLS de código aberto na plataforma Linux.

Durante a revisão bibliográfica obteve-se um entendimento mais aprofundado do protocolo LDP e do ambiente MPLS, oferecendo subsídios para a fase de especificação da solução de autenticação fim a fim.

A fase de desenvolvimento do protótipo foi subdividida em duas etapas:

a) seleção de uma implementação MPLS de código aberto;

b) realização do desenvolvimento do protótipo para validar em um ambiente real a solução de autenticação proposta.

A plataforma Linux foi escolhida pois além de ser um ambiente bastante difundido e amigável, possui a maior gama de projetos relacionados ao LDP, por exemplo [LEU, 2000].

1.4 Organização do Trabalho

Este trabalho é constituído por 7 capítulos. O capítulo 2 apresenta de forma sucinta os principais conceitos, características e o funcionamento básico do MPLS e descreve como ocorre o processo de distribuição de etiquetas. O capítulo 3 apresenta e discute as principais características e o funcionamento do protocolo LDP. O capítulo 4 faz considerações sobre segurança, autenticação e integridade dentro do ambiente LDP e apresenta trabalhos relacionados. No capítulo 5, a proposta de autenticação fim a fim para o LDP é descrita, e, no capítulo 6, os resultados de implementação obtidos são relatados. O capítulo 7 apresenta a análise dos resultados e as conclusões deste trabalho.

No ANEXO 3 é descrito como ocorre o funcionamento da atual forma de autenticação, baseada em MD5, definida para o LDP [ANDERSSON, 2001].

2 MPLS (MULTI-PROTOCOL LABEL SWITCHING)

2.1 Introdução

No roteamento IP tradicional todos os dados são tratados da mesma forma pois não há classes de serviços diferenciados. É utilizado o roteamento baseado no menor esforço (*best effort*), sendo que o algoritmo de roteamento mais usado é o *hop-by-hop.* Os pacotes são encaminhados um a um através de roteadores, que tomam a decisão de roteamento, baseado na inspeção do cabeçalho IP de cada pacote individualmente. Assim existem vários problemas com relação ao roteamento IP tradicional:

- roteadores situados no núcleo ou periferia de *backbone* acabam armazenando um grande número de entradas nas tabelas de roteamento;
- as buscas nas tabelas de roteamento são demoradas pois as entradas não têm tamanho fixo: procura-se o maior prefixo de endereço que coincide com o endereço de destino do pacote (*Longest Match*);
- o processo de roteamento é realizado em cada roteador e para cada pacote que chega no roteador. Esse fato acaba gerando atraso na propagação dos datagramas;
- este tipo de roteamento leva, às vezes, a um desequilíbrio na rede, com o tráfego se concentrando em alguns roteadores, congestionando esta área, enquanto outros roteadores ficam subtilizados;

Técnicas como Engenharia de tráfego[1] e Comutação de pacotes[2] são alternativas para tentar amenizar estes problemas nas redes IP. O MPLS utiliza ambas as tecnologias.

---

[1] Engenharia de tráfego são técnicas empregadas para otimizar o tráfego das redes e contornar situações de congestionamento.

Com evolução da informática e das redes de computadores surgiu a necessidade de se transmitir não só dados de texto, mas também informações multimídia, como áudio e vídeo. Tendo em vista que a tecnologia de roteamento das atuais redes IP possui vários problemas, principalmente a nível de roteamento em *backbone* (considerando um universo como a internet), o que praticamente inviabiliza aplicações multimídia, criou-se a necessidade de melhorias nas rede IP e assim surgiram algumas propostas neste sentido.

Inicialmente em sua maioria estas propostas eram baseadas na tecnologia ATM. A idéia principal era deixar de lado o roteamento com a busca baseada no prefixo mais longo (*longest match*) em vastas tabelas de roteamento e passar a utilizar uma busca exata em tabelas indexadas de porte bem menor, como etiquetas (*labels*). A cada pacote era anexado uma etiqueta (de tamanho fixo) e os roteadores tomavam decisões baseados nas etiquetas e não mais no endereço do destino. Exemplos destas tecnologias são *Tag Switch* (Cisco), *ARIS* (IBM) e *Cell Switched Router* (Toshiba). Porém, como tratava-se de tecnologias proprietárias incapazes de interoperar, surgiu então a necessidade de um modelo padrão de comutação por etiquetas, modelo que veio a ser o MPLS.

2.2 Características e Funcionamento do MPLS

O MPLS *(Multiprotocol Label Switchin*g) é um Framework definido pelo IETF *(Internet Engineering Task Forc*e) que proporciona designação, encaminhamento e comutação eficientes de fluxos de tráfego através da rede. É uma técnica de comutação de pacotes baseada em etiquetas.

É multiprotocolo pois pode ser aplicado a todos os protocolos de nível 3 (camada de rede), embora o interesse da comunidade esteja voltado a usar o MPLS com o tráfego IP. O MPLS é neutro quanto a tecnologia de rede, ou seja, pode ser implantado sobre redes ATM, Frame Relay, Ethernet, X.25 e outras.

---

[2] Comutação de pacotes são técnicas de chaveamento de pacotes baseados em circuitos virtuais geralmente implementados em hardware, assim tornam o processo mais ágil, por exemplo é a caraterística básica dos *swicthes*.

A informação em uma rede MPLS é designada a uma determinada classe de serviço e os dados são encaminhados através de caminhos pré-estabelecidos (LSPs), sendo feita apenas a comutação e não o roteamento dos pacotes. O MPLS é uma tecnologia utilizada em *backbones* e se destina a tratar problemas atuais nas redes de computadores como velocidade, escalabilidade, garantia de qualidade de serviços (QoS) e engenharia de tráfego (TE). Estes dois últimos assuntos tem sido atualmente o foco principal do interesse pelo MPLS.

O funcionamento do MPLS é baseado na identificação dos pacotes que são transportados por uma mesma rota ou, em outras palavras, que pertençam à mesma classe de encaminhamento ou FEC *(Forward Equivalence Clas*s). A identificação da FEC à qual um pacote pertence é realizada pelo roteador de borda de entrada (ingresso) do *domínio MPL*S[3]. Cada pacote FEC recebe uma etiqueta (*label*) de um determinado roteador, neste caso de ingresso (LER – *Label Edge Router*). Os pacotes são encaminhados através de um caminho comutado por etiquetas (LSP – *Label Switch Path*), formado por roteadores de comutação por etiquetas (LSRs – *Label Switch Routers*). Cada LSR toma decisões de encaminhamento baseado apenas na etiqueta do pacote. A cada nó de rede (*hop*) o LSR retira a etiqueta existente e aplica uma nova etiqueta que diz ao próximo nó de rede como encaminhar este pacote.

É importante ressaltar que a etiqueta MPLS pode assumir as mais variadas formas, dependendo do mecanismo de encaminhamento empregado na sua implementação, podendo ser representado por um par (VPI/VCI) no ATM, um (DLCI) no Frame Relay ou mesmo um novo cabeçalho *(shim-heade*r, como é chamado) acrescentado aos pacotes IP, caso sejam utilizadas diretamente tecnologias de nível 2 como a Ethernet ou PPP *(Point-to-Point Protoco*l), por exemplo.

Independente do formato, uma etiqueta MPLS sempre tem significado local em cada roteador, devendo a associação das mesmas a FECs ser negociada entre os LSRs vizinhos (adjacentes), o que pode ser feito de forma estática ou dinâmica. Na associação

---

[3] Um domínio MPLs consiste de um conjunto de roteadores MPLs (LSRs) que pertencem a uma administração comum.

estática, os mapeamentos são configurados manualmente pelo administrador do domínio MPLS em contraste com a associação dinâmica, onde um protocolo de sinalização é empregado com o mesmo propósito. O LDP é o protocolo recomendado pelo IETF para troca de associações de etiquetas-FECs em provedores MPLS, porém atualmente outros protocolos têm sido estendidos de forma a permitir a distribuição de etiquetas MPLS como é o caso do BGP (*Border Gateway Protocol)* [REKHTER, 2001] e do RSVP (*Resource Reservation Protocol)* [AWDUCHE, 2001].

Na borda de saída do *domínio MPLS,* cabe aos roteadores de borda de saída (LERs de egresso) a retirada das etiquetas comutadas e a posterior entrega dos pacotes originais ao destino.

Algumas vantagens do MPLS podem ser ressaltadas [MAGALHÃES, 2001]:

- a transferência da comutação da camada 2 para a camada 3, o que permite a criação de circuitos virtuais, em analogia as redes ATM e Frame Relay por exemplo;
- orientação a conexão em redes IP;
- CoS (Classes de Serviço). Atualmente no IP tradicional é empregado apenas o menor esforço (*best effort*);
- QoS (Qualidade de Serviço);
- facilidade para implantar Engenharia de Tráfego (TE);
- maior velocidade e menor complexidade de decisões de encaminhamento nos roteadores;
- suporte a VPNs, ou seja, suporte nativo às tecnologias de túnel e hierarquia, através do uso de pilhas de etiquetas;
- eliminação de múltiplas camadas;
- coexistência com as atuais redes IPs, possibilitando assim uma implantação incremental.

Apesar de concebido inicialmente para aumentar a velocidade da comutação IP, o interesse atual pelo MPLS está focado principalmente na facilidade de implantar

Engenharia de Tráfego(TE) e garantia de qualidade de serviços (QoS); na sua possibilidade de extensão para controle das futuras redes óticas de transporte (*Multiprotocol Lambda Switching*) e na possibilidade de coexistência com as atuais redes IP, viabilizando assim uma atualização incremental.

2.3 Conceitos Básicos no MPLS

Para uma compreensão do MPLS, dentre os muitos conceitos existentes, podemos citar alguns conceitos básicos principais: FEC, LER, LSR, LSP, Etiqueta (*label*) , Pilha de Etiquetas e Pacote Etiquetado.

2.3.1 FEC (*Forward Equivalence Class*)

É um grupo de pacotes que é encaminhado segundo um mesmo critério. A colocação de uma etiqueta em um pacote significa atribuir este pacote a uma determinada FEC (Figura 1).

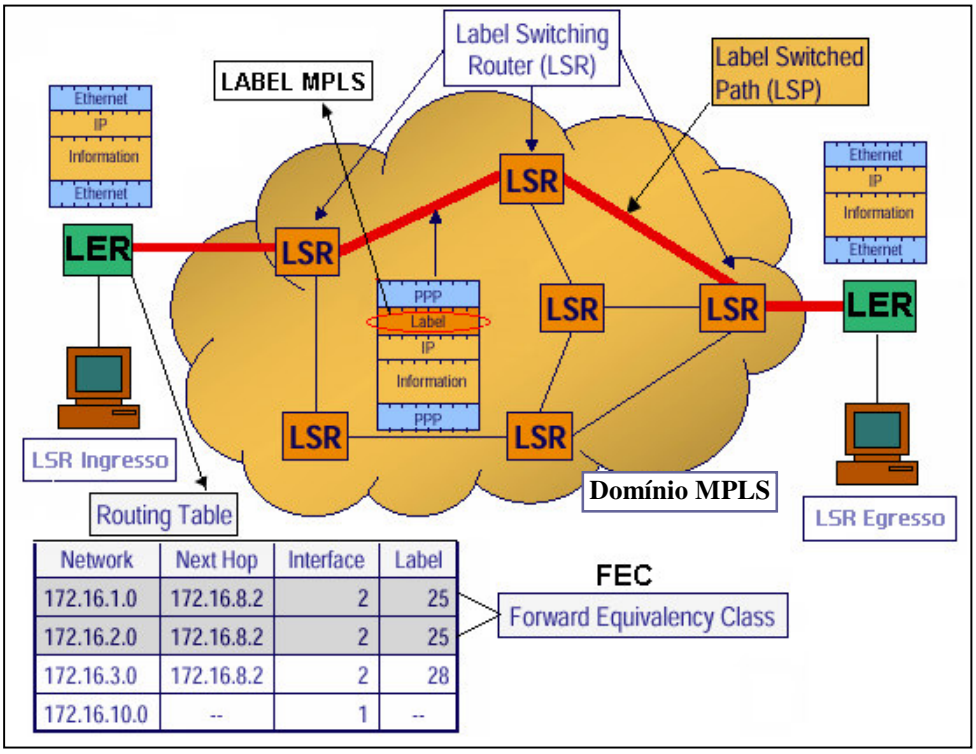


Figura 1. Conceitos básicos no MPLS (FEC, LER, LSR e LSP).

Há atualmente dois tipos de elementos FEC definidos para o MPLS: prefixos de endereços (por exemplo um endereço IP de rede, como 192.168.1.0/24) e endereços completos (por exemplo um endereço IP de host, como 192.168.1.1/32). O conceito de FEC é importante pois, como as FECs agrupam pacotes baseados em um critério comumo, é possível criar FECs para satisfazer requisitos de QoS ou de Engenharia de Tráfego.

2.3.2 LER (*Label Edge Router*)

LERs são LSRs situados nas bordas da rede MPLS (Figura 1). Eles ligam diversas sub-redes (Ethernet, Frame Relay, ATM) à rede MPLS, na verdade, são LSRs que tem a capacidade de fazer fronteira com outras redes. São responsáveis pela inserção e remoção de pilhas inteiras de etiquetas dos pacotes para o tráfego que entra e sai da rede MPLS.

Os LERs de Ingresso realizam a rotulagem (atribuição de etiquetas) das novas FECs (novas rotas IP inseridas da tabela de roteamento IP) que ingressam no domínio MPLS. Os LERs de Egresso retiram as etiquetas MPLS dos pacotes na saída do domínio MPLS e encaminham o mesmo para a rede conectada (por exemplo uma rede IP, ATM ou Frame Relay).

2.3.3 LSR (*Label Switching Router*)

LSRs são roteadores MPLS de comutação por etiquetas (Figura 1). São equipamentos situados no núcleo da rede MPLS e sua função é encaminhar os pacotes baseados apenas na etiqueta de cada pacote. São responsáveis por fazer a troca, retirada ou inserção de etiquetas na pilha do pacote.

Ao receber um pacote, o LSR troca a etiqueta atual por uma nova etiqueta correspondente ao próximo nó de rede no caminho, através de uma busca pela FEC correspondente em sua tabela de etiquetas local. Depois encaminha o pacote para o próximo roteador do caminho e assim por diante.

Os LSRs atuam junto aos LERs (LSRs das bordas do domínio MPLS) na troca de mensagens para requisição ou atribuição de etiquetas para manutenção dos LSPs.

2.3.4 LSP (*Label Switch Path*)

Um LSP consiste em um caminho comutado por etiqueta, ou seja, um caminho que atravessa uma seqüência de LSRs, estabelecido entre uma origem e um destino (Figura 1). O LSP é unidirecional, portanto é preciso ter dois LSPs para uma comunicação entre duas entidades. Através de um LSP trafegam pacotes de uma mesma classe de serviço (CoS), que compartilham o mesmo destino.

Antes de estabelecer o LSP, uma rota deve ser estabelecida, tarefa que é realizada por protocolos de roteamento convencionais ou roteamento baseado em restrições. Após o estabelecimento do LSP o caminho fica pré-definido e os pacotes que pertencem a ele não precisam mais ser roteados, eles serão apenas comutados com base nas suas etiquetas. As etiquetas são distribuídas entre os LSRs no momento do estabelecimento dos LSPs. A própria estrutura de pilha das etiquetas permite que se tenha vários níveis de LSPs na rede, que são chamados de túneis. Para cada LSP existe um roteador de início e de fim, denominados LSRs de ingresso e egresso respectivamente.

2.4 Roteamento de LSPs

O roteamento de LSPs diz respeito ao método utilizado para seleção da rota de destino durante o estabelecimento de um LSP. A arquitetura MPLS prevê duas opções para o estabelecimento de rotas, o roteamento *hop-by-hop* e o roteamento explícito.

2.4.1 Roteamento *hop-by-hop*

Nas redes IP, em geral, os pacotes seguem o menor caminho entre a fonte e o destino. Esse caminho vai sendo percorrido através de cada roteador, onde é selecionada a entrada da tabela de roteamento mais adequada de acordo com o destino especificado no cabeçalho do pacote (*Longest Matc*h).

O roteamento *hop-by-hop*, via LDP por exemplo, utiliza as mesmas entradas das tabelas de roteamento IP dos LSRs, após estas serem criadas por um determinado protocolo de roteamento que pode ser OSPF, BGP, RIP ou outros. A partir daí, são atribuídas uma ou mais etiquetas de entrada e saída para cada prefixo de endereço existente na tabela de roteamento. Em seguida, essas etiquetas serão distribuídas para os vizinhos de cada roteador (LSR).

Uma das vantagens de se usar o MPLS com o roteamento *hop-by-hop* tradicional é que só existirá roteamento de pacotes de dados na borda do domínio MPLS, pois, após o estabelecimento dos LSPs, o encaminhamento de pacotes será realizado baseado nas etiquetas MPLS.

2.4.2 Roteamento Explícito

O roteamento será baseado em um caminho pré-definido. Este caminho pode ser definido por um protocolo de roteamento explícito, como o CR-LDP por exemplo, ou por configuração manual, por exemplo o administrador do domínio MPLS cria rotas estáticas nos LSRs.

No caso de LSPs roteados explicitamente, os LSRs não são autônomos na escolha do próximo *hop*. Na realidade, um único LSR, normalmente o LSR de ingresso ou o LSR de egresso, especifica todos (ou parte) dos LSRs que compõem a rota do LSP. Caso sejam especificados todos os LSRs da rota, diz-se que se trata de um LSP roteado explicitamente de forma completa. Em situações onde são especificados somente alguns dos LSRs que compõem a rota do LSP, diz-se que se trata de um LSP roteado explicitamente de forma “fraca”.

Normalmente, o roteamento explícito pode ser realizado por configuração manual (pelo administrador do domínio MPLS) ou dinamicamente, neste último caso baseado em protocolos de roteamento que trabalham com o conceito de estado dos enlaces (*link state*), como o OSPF por exemplo. O maior interesse no roteamento explícito está na possibilidade de implementação da engenharia de tráfego. Apesar de ser possível o roteamento IP na origem (*source routing*) em redes IP convencionais, o mesmo nunca foi empregado na engenharia de tráfego pela sua ineficiência. Esta ineficiência reside no

fato do roteamento IP, na origem, necessitar que cada datagrama IP transporte sua rota no campo opções do pacote IP, aumentando o tamanho e o tempo de processamento do datagrama. Como o roteamento explícito em MPLS estabelece a rota quando da criação do LSP, datagramas trafegando por um LSP não necessitam transportar sua rota, pois a mesma está definida pela etiqueta. Como conseqüência, o roteamento explicito em MPLS é muito mais eficiente pois elimina a necessidade do transporte e manipulação de opções nos datagramas IP, mantendo seu processamento limitado à camada de enlace no interior do domínio MPLS.

Os principais protocolos de roteamento dinâmico explícito no MPLS são o CR-LDP e o RSVP-TE.

2.5 Distribuição de Etiquetas no MPLS

2.5.1 Etiquetas MPLS

Uma etiqueta é um identificador curto, de tamanho fixo e significado apenas local ao domínio MPLS, que é usado para identificar uma FEC, portanto está sempre associado a uma FEC. Esta associação é empregada para:

- decidir que etiqueta atribuir ao datagrama no LSR de ingresso;
- determinar a rota de um LSP nos LSRs de ingresso e núcleo;
- decidir se o LSR é o de egresso.

MPLS faz uso de etiquetas para associar datagramas IP a LSPs. O cabeçalho MPLS deve ser posicionado depois de qualquer cabeçalho da camada 2 (enlace) e antes de um cabeçalho de camada 3 (transporte), por esse motivo é também denominado "*shim header*", ou seja, cabeçalho de camada 2,5. Seu tamanho é definido em 4 octetos e seu formato está ilustrado na Figura 2.

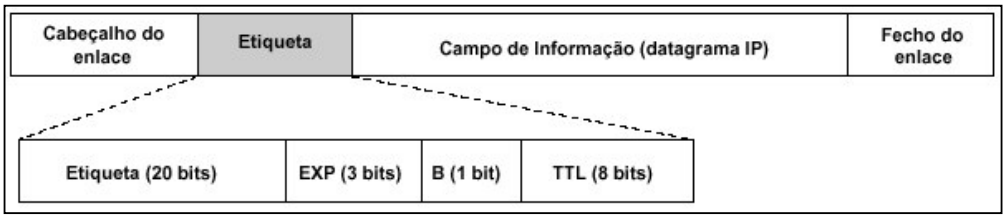


Figura 2. Uma Etiqueta MPLS posicionada dentro de um pacote de enlace.

- o campo *Etiqueta* (20 bits) carrega o valor atual do etiqueta MPLS;
- o campo *EXP* (3 bits) pode afetar o enfileiramento e algoritmos de descarte aplicados ao pacote enquanto ele é transmitido pela rede. É um campo que ainda não está totalmente definido;
- o campo *B* (*Stack Bit)* suporta uma pilha hierárquica de etiquetas. O Valor 1 indica base da pilha e os demais níveis são atribuídos em 0;
- o campo *TTL* (8 bits) fornece funcionalidades de TTL (*Time to Live)* do IP convencional ao MPLS. Especifica um limite de quantos *hops* o pacote pode atravessar.

2.5.2 Pilha de Etiquetas

A comutação por etiquetas foi projetada para ser usada em redes de grande porte. Nesse sentido, a arquitetura MPLS permite que um pacote carregue várias etiquetas organizados na forma de uma pilha (a última etiqueta inserida é a primeira a ser removida). A Figura 3 mostra uma pilha de etiquetas em um pacote na rede.

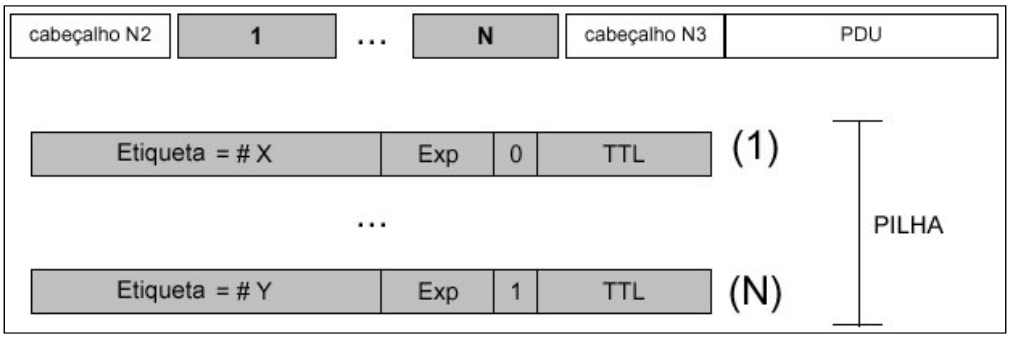


Figura 3. Exemplo de uma pilha de etiquetas em um pacote da rede. Observe que o campo "Stack Bit" do último pacote possui o valor 1.

A questão fundamental envolvendo a pilha de etiquetas é que a decisão de encaminhamento realizada nos LSRs é sempre baseada na etiqueta que se encontra no topo da pilha. Um dos aspectos fundamentais da pilha de etiquetas consiste na possibilidade de definir uma hierarquia de etiquetas, hierarquia esta que possui implicações importantes do ponto de vista do roteamento e do encapsulamento (túneis) de informações. A Figura 4 [MAGALHÃES, 2001] ilustra dois domínios MPLS hierarquizados onde um LSP estabelecido no domínio externo atravessa o domínio interno.

Um datagrama, ao ingressar no domínio externo, recebe uma etiqueta formando uma pilha com um único elemento. Este datagrama, ao ingressar no domínio interno, será comutado através de um LSP diferente do original, e para tanto, recebe uma nova etiqueta correspondente ao LSP interno. Esta nova etiqueta é acrescentada no topo da pilha, preservando-se assim a etiqueta com a qual o datagrama ingressou no domínio interno. No domínio interno, o datagrama é comutado com base na etiqueta do topo da pilha. O penúltimo LSR de egresso do domínio interno remove a etiqueta do topo da pilha, encaminhando o datagrama para o domínio externo com a etiqueta inicial. A partir daí, a etiqueta original passa a ser empregada no domínio externo. Deve-se observar ainda neste cenário que:

a) o LSP externo é encapsulado (túnel) através do LSP interno;

b) a retirada da etiqueta da pilha se dá no penúltimo[4] LSR do domínio;

c) o LSP interno é visto como um enlace para os LSRs R2 e R3.

d) A pilha de etiquetas não degrada o desempenho da comutação por etiquetas;

e) múltiplos níveis de hierarquias e encapsulamento (túneis) podem ser obtidos.

---

[4] A remoção da etiqueta no penúltimo ou último LSR é irrelevante. Porém no penúltimo LSR é mais eficiente pois assim o último LSR economiza uma busca em sua tabela de etiquetas.

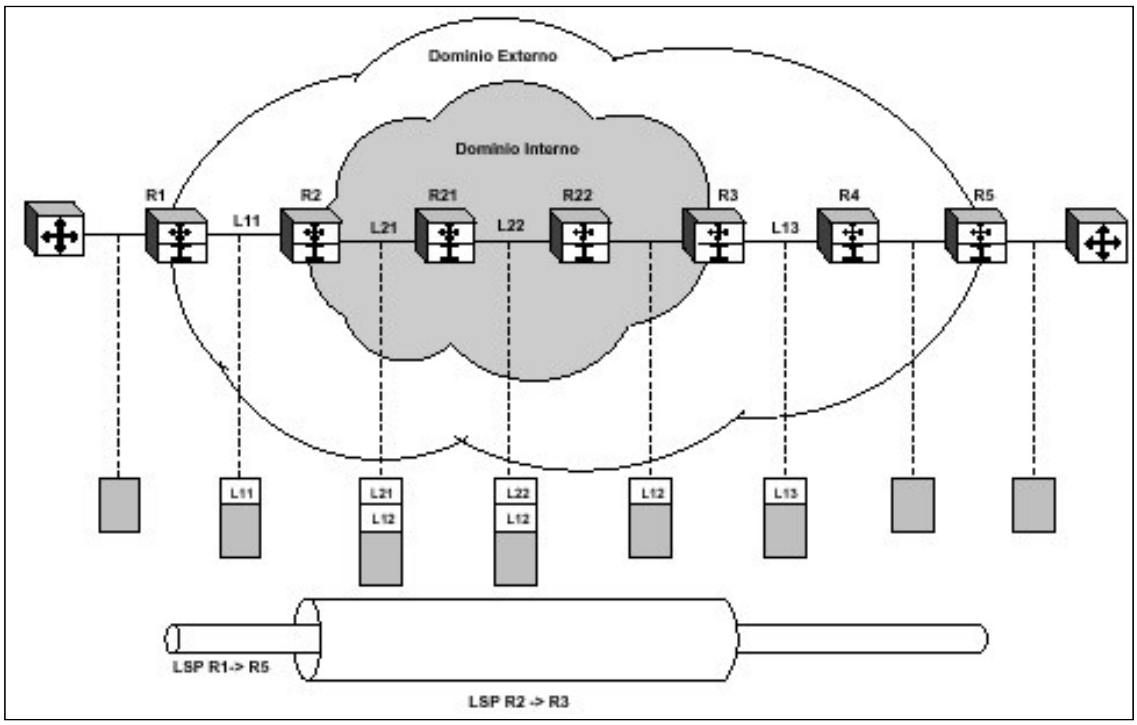


Figura 4. Pilhas de etiquetas: Túneis e Hierarquias [MAGALHÃES, 2001]

2.5.3 Pacote Etiquetado

Um pacote Etiquetado é um pacote que contém uma pilha de etiquetas associada, ou seja, já passou por um roteador MPLS e foi devidamente etiquetado ou codificado para o ambiente MPLS. Um pacote com uma pilha de etiquetas que contém apenas etiquetas nulas, continua sendo um pacote etiquetado. Uma pilha de etiquetas nulas pode ser usada, por exemplo, quando se quer diferenciar uma célula ATM etiquetada de uma célula não etiquetada.

2.5.4 Protocolos de Distribuição de Etiquetas

A arquitetura MPLS não impõe uma estratégia específica para distribuição de etiquetas, ao contrário, prevê a existência de diferentes estratégias e protocolos para esta finalidade. Existem basicamente duas estratégias para distribuição de etiquetas em uma rede MPLS:

a) adicionar capacidade de distribuição de etiquetas a um protocolo já existente;

b) conceber um novo protocolo para distribuição de etiquetas.

Existem propostas que contemplam as duas estratégias descritas acima.

Uma solução que emprega a primeira estratégia (a) consiste na extensão do protocolo de roteamento BGP (*Border Gateway Protocol*) [REKHTER, 2001] para MPLS. Nesta solução, etiquetas são distribuídas anexadas as mensagens BGP. Outra solução nesta linha consiste em fornecer capacidade de distribuição de etiquetas ao protocolo RSVP (*Resource Reservation Protocol*) [AWDUCHE, 2001]. Nesta solução, novos objetos são definidos para mensagens RSVP. Estes objetos permitem a requisição e resposta de associações etiqueta-FEC, além de outras mensagens de controle (por exemplo, desativação de LSPs).

A segunda estratégia é usada pelo protocolo LDP (*Label Distribution Protoco*l) [ANDERSSON, 2001]. O LDP foi concebido exclusivamente para a distribuição de etiquetas em redes MPLS. A diferença fundamental entre o RSVP e o LDP no tocante a distribuição de etiquetas é que uma associação etiqueta-FEC deve ser confirmada periodicamente pelo RSVP, caso contrário a associação é desfeita por *timeout* enquanto no protocolo LDP, uma vez estabelecida a associação, a mesma se mantém até que seja explicitamente desfeita ou até que a relação de vizinhança entre os LSRs pares envolvidos seja encerrada.

2.5.5 Estratégias de Distribuição de Etiquetas

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).

As etiquetas são distribuídas entre os LSRs e toda etiqueta tem sempre uma FEC associada. A forma de distribuição de etiquetas denomina-se distribuição via *downstream* (ou seja, o LSR posterior gera e atribui a etiqueta para o LSR anterior). Esta forma de distribuição especifica que, dado um sentido de um fluxo, qualquer

associação etiqueta-FEC é sempre criada pelo LSR posterior (ou seja, via *downstream*) e distribuída para o LSR anterior (ou seja, via *upstream*) (Figura 5).

Cada LSR mantém uma Base de Informação de etiquetas, ou LIB (*Label Information Base*) (Figura 5). A LIB contém pelo menos cinco colunas: a FEC, a etiqueta de entrada, a interface de entrada, a etiqueta de saída e a interface de saída.

O processamento de etiquetas difere entre os LSRs dependendo de sua posição dentro do domínio MPLS (ingresso, egresso ou núcleo). LSRs de ingresso são responsáveis por etiquetar pacotes quando estes coincidem com os critérios de determinada FEC. Esta FEC determina uma entrada na LIB que estipula a etiqueta que o datagrama irá receber, bem como a interface de saída. Nos LSRs do núcleo, o datagrama será encaminhado exclusivamente com base na etiqueta. Ao receber um datagrama etiquetado em uma determinada interface o par "etiqueta-interface" determina uma entrada na LIB. O datagrama que tem sua etiqueta comutada é encaminhado à interface de saída. Finalmente, no LSR de egresso, a etiqueta é removida e o datagrama passa a ser roteado de forma convencional (*"hop-by-hop"*) pela camada de rede (por exemplo, via roteamento IP tradicional).

A arquitetura MPLS permite a um LSR requisitar explicitamente ao seu par, para uma FEC em particular, uma atribuição de uma etiqueta. Este método é conhecido como "Distribuição Sob Demanda". A arquitetura MPLS permite também que um LSR distribua etiquetas para outros LSRs que não as tenham requisitado explicitamente, método este conhecido como "Distribuição Não Solicitada".

a) Distribuição Não Solicitada

Neste método, um LSRx envia para LSRy uma atribuição de etiqueta para uma FEC sem que LSRy a solicite explicitamente. LSRx o faz, por exemplo, quando recebe uma ou mais atribuições de um novo LSR par válido, para uma determinada FEC e quer distribuir as atribuições imediatamente para seus pares via *upstream*. Na Figura 5, o LSR3 detecta uma nova FEC (prefixo 64.12); LSR3 envia para LSR2 a atribuição de etiqueta para a FEC; LSR2 aceita e então repete o procedimento para LSR1. Repare

que a Figura 5 ilustra o sentido da conexão (*upstream/downstream*) e mostra as entradas correspondentes na LIB de cada LSR.

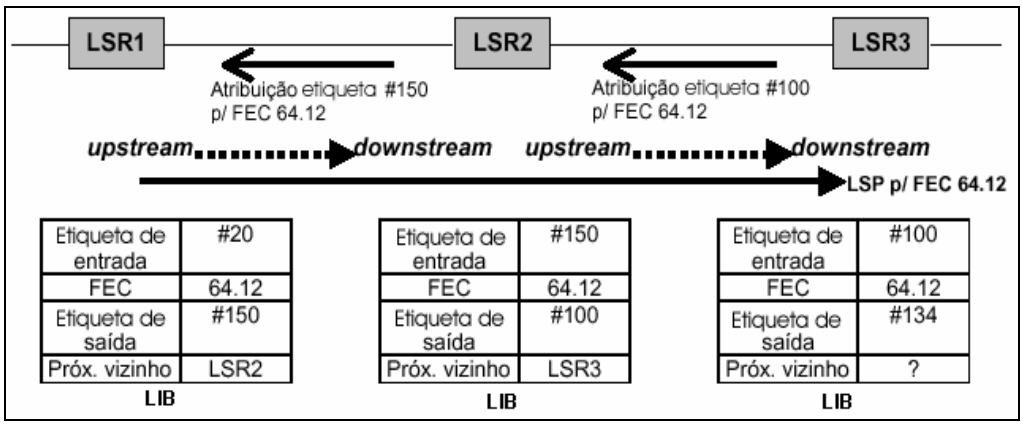


Figura 5. Distribuição Não Solicitada

b) Distribuição Sob Demanda

Um LSRx envia para LSRy uma solicitação de atribuição de etiqueta para uma dada FEC, após detectar este último como vizinho válido para esta FEC. Esta detecção pode ser realizada se LSRy enviar um prefixo para LSRx, tornando-se, então, um vizinho válido para este roteador. Neste caso, o requisitante pode ser um LER de entrada ou então está enviando esta requisição como resultado de uma outra requisição que chegou a ele. Na Figura 6, o LSR1 requisita ao LSR2 uma atribuição de etiqueta para a FEC 64.12; LSR2 repete o procedimento para LSR3; LSR3 tem um mapeamento e envia para LSR2; LSR2 tem o mapeamento e envia para LSR1.

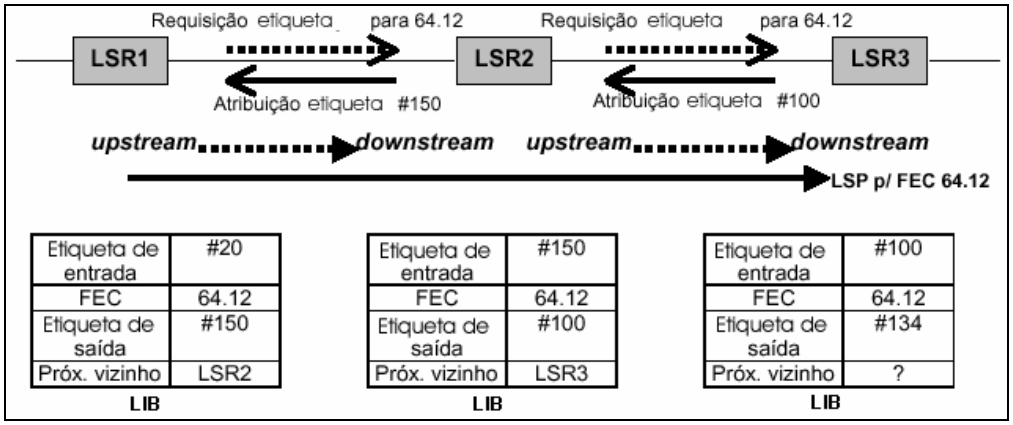


Figura 6. Distribuição Sob Demanda

c) Modos de Controle e Modos de Retenção de Etiquetas

A distribuição de associações Etiqueta-FEC pode ter controle ordenado ou independente.

No controle ordenado, o LSR distribui uma associação para o LSR *upstream* apenas quando já possui uma associação de uma etiqueta para a mesma FEC suprida pelo LSR *downstream*, ou quando ele for o LSR de egresso para esta FEC. Nesta forma, durante o estabelecimento de um LSP, as associações são sempre distribuídas a partir do LSR de egresso em direção ao LSR de ingresso.

No controle independente, um LSR pode distribuir associações no momento que desejar. Por exemplo, operando segundo a estratégia de Distribuição Sob Demanda, um LSR no modo de Controle Independente pode responder uma requisição de associação para o LSR *upstream* antes de propagar a requisição para o LSR *downstream*.

O modo de retenção das associações pode ser Conservativo ou Liberal. Esta classificação se aplica apenas quando a estratégia de distribuição de etiquetas for Sem Solicitação. No modo conservativo, o LSR mantém apenas as associações etiqueta-FEC recebidas de um LSR *downstream* quando este LSR é 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.

Para as redes que utilizam identificadores de conexão (ex. etiquetas), tal como redes ATM e Frame Relay, as etiquetas são recursos limitados. Para estas redes, a distribuição de etiquetas deve minimizar o desperdício destes recursos. Portanto, a estratégia de Distribuição Sob Demanda, com Controle Ordenado é a mais apropriada. De fato, esta forma de distribuição de etiqueta tem sido a mais empregada em redes MPLS.

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

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

Tabela 1. Resumo das Estratégias de Distribuição de Etiquetas

2.6 Conclusões do Capítulo

Este capítulo apresentou as características, conceitos básicos e o funcionamento do MPLS. Foi descrito como ocorre a distribuição de etiquetas, as estratégias possíveis, características e conceitos associados e foram realizadas considerações sobre os protocolos utilizados nesta tarefa.

3 LDP (*LABEL DISTRIBUTION PROTOCOL*)

3.1 Características e Funcionamento do LDP

O LDP é o protocolo que garante, por meio de trocas de mensagens entre os LSRs (usando TCP e UDP), que todas as suas tabelas de etiquetas (LIBs) estejam atualizadas e sem conflitos, a fim de manter corretamente os LSPs.

O protocolo LDP corresponde a um conjunto de procedimentos e mensagens que permitem aos LSRs estabelecerem LSPs através da rede. O LDP associa a cada LSP criado uma FEC para definição dos pacotes que serão mapeados naquele LSP específico. Cada FEC é especificada como um prefixo de endereço de qualquer comprimento ou um endereço IP completo.

LSRs que utilizam LDP para trocar informação sobre etiquetas e FECs são denominados Pares LDP *(LDP Peer*s) e, para este fim, estabelecem entre si uma Conexão LDP (*LDP Sessio*n). Para viabilizar a comunicação entre os pares LDP é primordial que ambos negociem e concordem no significado das etiquetas utilizadas para encaminhar o tráfego entre eles e através deles, parâmetros estes que são negociados no estabelecimento da sessão LDP.

A comunicação entre Pares LDP é realizada através da troca de mensagens. Para a identificação do LSR gerador da mensagem, utiliza-se os Identificadores LDP, que são compostos pelo Identificador do LSR, globalmente único, e pelo Espaço de Etiqueta *(Label Space*), que é definido dentro do escopo do LSR.

De forma geral, O LDP duplica as rotas (FECs) que o IP usa para propósitos de roteamento, transformando esta estrutura em etiquetas (*labels*). No IP são usados algoritmos de roteamento dinâmicos como BGP, OSPF, RIP e outros para computar e distribuir os pacotes sendo que cada roteador do caminho examina o destino do pacote e escolhe um novo link. No LDP o pacote segue exatamente o mesmo caminho do IP, porém, é atribuída uma etiqueta e um link quando um pacote com uma nova FEC é

roteado pela primeira vez no domínio MPLS. Quando o pacote chega no próximo *hop*, esta etiqueta é substituída por uma nova etiqueta, baseada na nova rota, e o pacote é encaminhado pelo link correspondente. Dessa maneira o pacote segue o mesmo caminho que seguiria através do roteamento IP, porém o seu cabeçalho nunca é examinado pelos roteadores intermediários, todo o roteamento é baseado nas etiquetas geradas e distribuídas pelo LDP, que são de tamanho fixo e pequenas (32 bits), esse fato fornece maior performance ao processamento.

3.2 PDUs (*Protocol Data Units*) LDP e TLVs (*Type-Length-Value)*

A troca das mensagens LDP se dá pelo envio de PDUs (*Protocol Data Units*) LDP sobre as sessões LDP estabelecidas. Cada PDU LDP é composta por um cabeçalho LDP seguido de uma ou mais mensagens LDP, encapsuladas em TLVs. As mensagens LDP transportadas numa mesma PDU não necessitam ter qualquer tipo de relação entre si. Por exemplo, uma PDU LDP pode transportar uma mensagem de LABEL REQUEST para um conjunto de FECs, outra mensagem de LABEL MAPPING para outro conjunto de FECs e ainda uma terceira mensagem de NOTIFICATION.

Uma PDU LDP é codificada da seguinte maneira (Figura 7 e Figura 8):

PDU-LDP = CABEÇALHO-LDP + [TLV, TLV, TLV, ...]

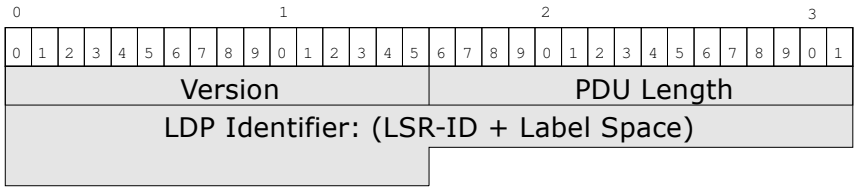


Figura 7. Formato do Cabeçalho LDP de uma PDU LDP

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 etiquetas 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 etiquetas (*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 Etiquetas (*Label Space*): O espaço de etiquetas 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 etiquetas. Quando usado por plataforma, significa que as interfaces do LSR podem compartilhar o mesmo espaço de etiquetas. Uma comunicação entre dois LSRs pode utilizar vários espaços de etiqueta, por motivos diversos, nesse caso é criada uma sessão TCP/LDP exclusiva para cada espaço de etiquetas em uso com o LSR par.

O LDP usa um **esquema de codificação** denominado ***Type-Length-Value*** **(TLV)** para codificar a maioria das informações de uma mensagem LDP (Figura 8).

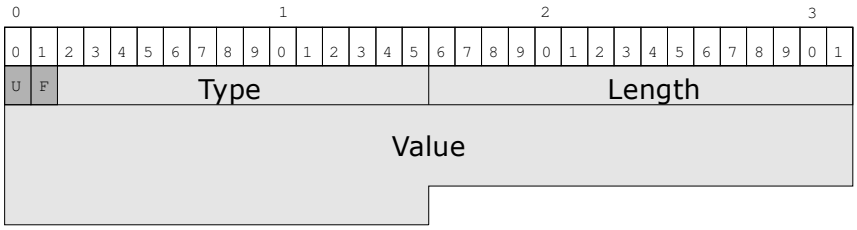


Figura 8. Formato de um TLV de uma PDU LDP

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 da mensagens LDP.

3.3 Categorias de Mensagens LDP

Quanto ao funcionamento do LDP, existem quatro categorias básicas de mensagens LDP:

- **mensagens de descoberta (*Discovery messages*):** são empregadas inicialmente para anunciar a presença de um LSR em uma rede e posteriormente para confirmar que o LSR continua presente;
- **mensagens de sessão (*Session messages*)** : são usadas para estabelecer, manter e encerrar sessões LDP entre dois LSRs;
- **mensagens de anúncio de etiqueta (*Advertisement messages*)**: são usadas para criar, modificar e excluir associações de etiquetas à FECs;
- **mensagens de notificação (*Notification messages*)**: são usadas para fornecer informações de estado da rede e para sinalizar erros.

3.4 Estabelecimento de Sessões LDP

Quando um LSR usa o LDP para anunciar mais de um espaço de etiquetas para um outro LSR, ele usa uma sessão LDP separada para cada espaço de etiquetas. No caso de serem requeridas múltiplas sessões LDP entre dois LSRs será criada uma conexão TCP (sessão de transporte) para cada sessão LDP.

A Figura 9, no quadro 1, mostra as fases de estabelecimento de uma sessão LDP.

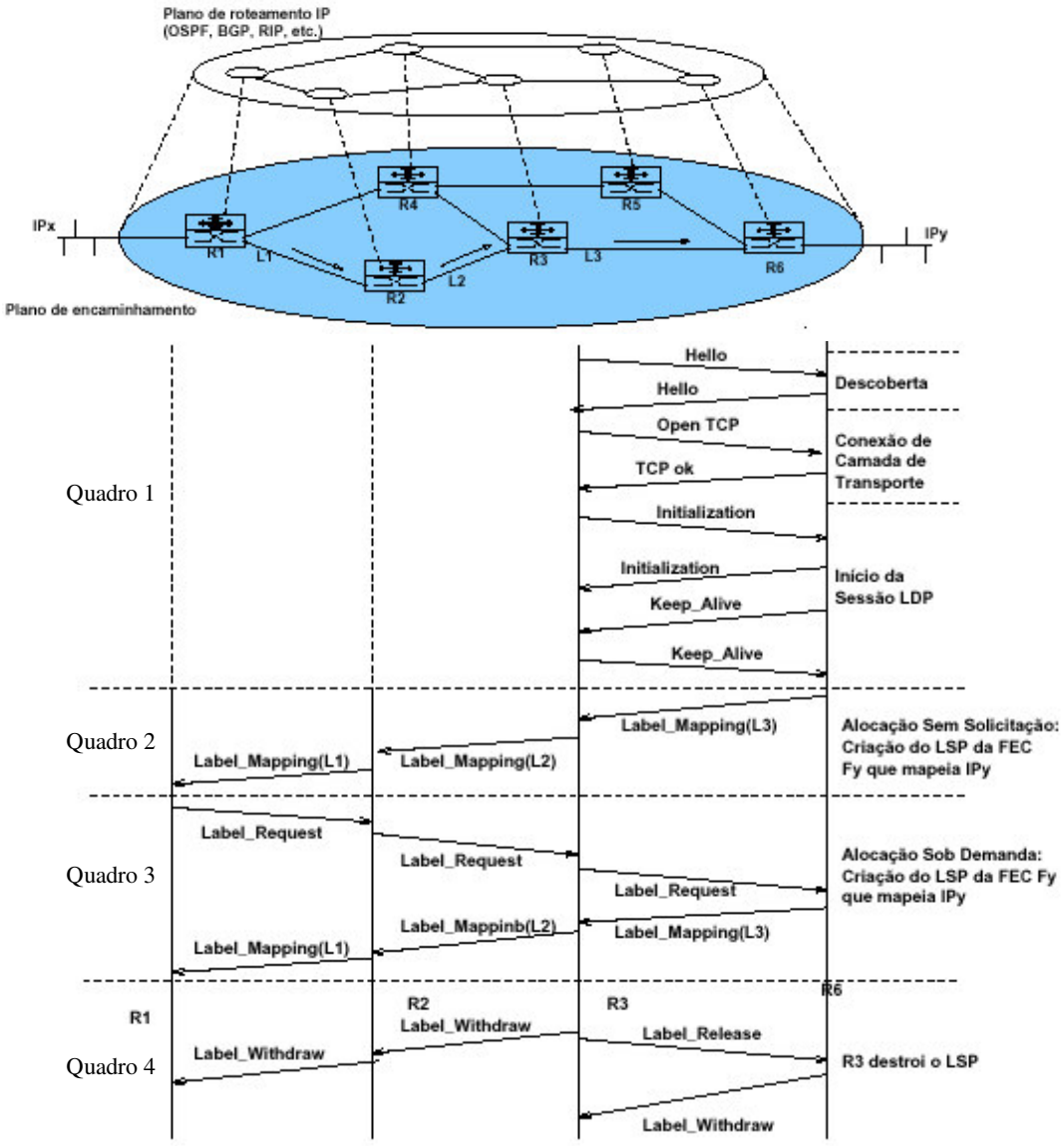


Figura 9. Exemplos de troca de mensagens LDP [MAGALHÃES, 2001]

O estabelecimento de uma sessão LDP se caracteriza pelas seguintes fases

a) descoberta do par LDP: Antes que dois LSRs adjacentes possam iniciar a troca de mensagens para a distribuição de etiquetas, eles devem estabelecer uma relação de vizinhança. Para descobrir se um LSR vizinho suporta o protocolo LDP, o LSR envia mensagens de descoberta (HELLO) aos vizinhos. Estas mensagens são enviadas como pacotes UDP, na porta 646 (LDP *well-know discovery port*), normalmente para o endereço de grupo *multicast* 224.0.0.2 (*all-routers multicast group*), caracterizando o mecanismo de Descoberta Básico do LDP (*Basic Discovery*).

Dois LSRs não diretamente conectados por enlace podem igualmente trocar mensagens HELLO, caracterizando o mecanismo de Descoberta Estendido do LDP (*Extended Discovery*) . Neste caso as mensagens de HELLO, denominadas "*Targeted Hellos",* são direcionadas a um endereço específico, via comunicação *unicast*. E do mesmo modo como no mecanismo de descoberta Básico estes "*Targeted Hellos" s*ão enviados em pacotes UDP para a porta 646 do endereço específico de destino. O recebimento de um HELLO do tipo "*Targeted Hello"* identifica uma "adjacência de hello" com um par LDP, alcançável no nível de rede, enquanto que um HELLO Básico identifica uma "adjacência de *hello*" com um par LDP, alcançável no nível de enlace.

Uma outra diferença relevante entre os mecanismos de descoberta do LDP é que o Básico é simétrico, enquanto que o Estendido é assimétrico. Quando um LSR inicia uma descoberta pelo mecanismo de descoberta Estendido com outro LSR, este LSR alvo é que decide se vai responder ou ignorar a requisição. Caso o LSR alvo escolher responder, ele faz isso enviando HELLOS do tipo "*Targeted Hellos"* periódicos ao LSR que iniciou a requisição.

Quando um LSR envia um HELLO ele anuncia o seu endereço de transporte. Ele pode fazer isso explicitamente incluindo um TLV de Transporte opcional, ou implicitamente omitindo o TLV de Transporte e usando o mesmo como endereço origem do HELLO. Outra informação relevante anunciada em uma mensagem de HELLO é o espaço de etiquetas que o LSR que iniciou a conexão (ativo) pretende usar.

b) **estabelecimento de uma conexão de transporte:** Uma vez descoberta a presença de outro LSR suportando LDP, os pares iniciam o estabelecimento de uma conexão de transporte TCP (porta 646). O estabelecimento da conexão de transporte

caracteriza-se pelo conceito de “papel” que cada LSR exerce na comunicação. O LDP define que o LSR com o maior endereço deve exercer o papel de “ativo” e o outro o papel de “passivo”. A comparação se dá entre os endereços de transporte dos dois LSRs como inteiros de 32 bits.

As informações do protocolo LDP necessitam ser transmitidas de forma confiável, razão pela qual o protocolo TCP é empregado.

c) **inicialização da sessão LDP:** Após o estabelecimento de uma sessão ao TCP para o transporte de PDUs LDP, um dos lados envia uma mensagem do tipo INITIALIZATION. Esta mensagem contém diversas informações quanto às características suportadas pelo LSR, dentre outras, versão do protocolo LDP, método de distribuição de etiquetas (Sob Demanda ou Não Solicitada), forma de destruição das etiquetas, tamanho máximo para PDUs e intervalo de mensagens *Keep Alive*. Após estabelecida a sessão LDP os LSRs tornam-se pares LDP.

d) **fase ativa**: Uma vez estabelecida a sessão LDP, os dois LSRs encontram-se na fase ativa para solicitação e distribuição de etiquetas.

3.5 Manutenção e Encerramento das Sessões LDP

A manutenção e o encerramento das sessões LDP tem dois contextos (a Figura 10 mostra estas fases de manutenção e encerramento):

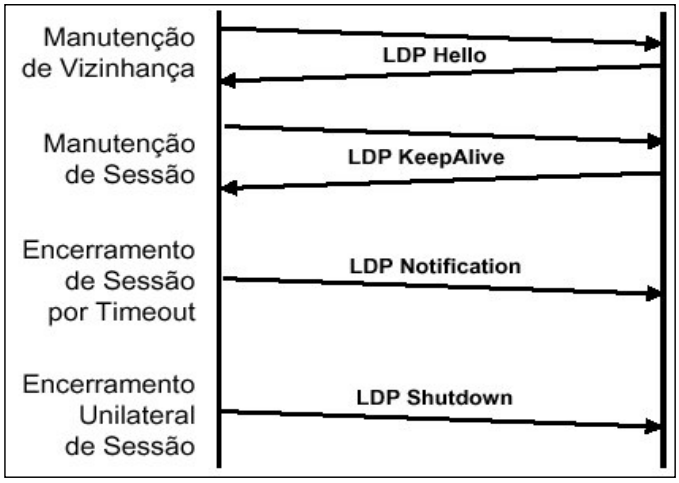


Figura 10. Manutenção e encerramento de uma sessão LDP

a) **Manter a relação de vizinhança:** (adjacências de HELLO) o LDP utiliza os as mensagens regulares de HELLO (iniciadas pelo mecanismo de Descoberta do LDP) recebidas, para identificar a intenção do par LDP em utilizar e continuar usando um espaço de etiquetas acertado. Assim 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 etiquetas 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).

b) **Manter a sessão LDP**: as sessões LDP são mantidas através de mensagens periódicas do tipo KEEP_ALIVE, em 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.

3.6 Gerência de Etiquetas

O protocolo LDP define mensagens para criação, distribuição e exclusão de associações Etiqueta-FEC (a Figura 9, nos quadros 2, 3 e 4, mostra exemplos de mensagens de gerência de etiquetas).

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

- *hop count*: mostra 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 laços. Se um LSR recebe uma mensagem LABEL REQUEST e faz parte do *path vector* desta mensagem, então a mensagem percorreu um laço (*loop*).

Ao receber uma mensagem LABEL REQUEST, o LSR receptor aloca uma etiqueta para a FEC. Caso o LSR seja o de egresso para a FEC, a etiqueta é prontamente retornada para o LSR requisitante. Caso contrário, se o LSR opera no modo ordenado, a requisição é propagada para o LSR que é o próximo *hop* para a FEC. Quando esta requisição for respondida pelo próximo *hop* (via *downstream*) a etiqueta alocada é inserida na LIB e uma etiqueta correspondente é gerada e retornada para o LSR requisitante. No modo independente, a etiqueta em geral é prontamente retornada, mesmo se o LSR não for o LSR de egresso para a FEC.

Para **distribuir** uma associação Etiqueta-FEC uma mensagem LABEL MAPPING é empregada. Esta mensagem contém duas informações obrigatórias: a FEC e a etiqueta atribuída. A mensagem LABEL MAPPING é gerada em resposta a uma mensagem LABEL REQUEST se a estratégia de distribuição for Sob Demanda. Caso opere segundo a estratégia de distribuição Não Solicitada, um LSR pode gerar uma mensagem LABEL MAPPING sem ter recebido uma requisição. Opcionalmente uma mensagem LABEL MAPPING pode conter mais três TLVs:

- *hop count* e *path vecto*r: com as mesmas funções nas mensagens LABEL REQUEST, conforme explicado acima;
- *message ID*: usado como identificador de requisição se em resposta a um LABEL REQUEST;

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

- LABEL WITHDRAW: mensagem enviada a um LSR via *upstream* para invalidar uma associação Etiqueta-FEC atribuída pelo emissor. Esta mensagem é gerada quando o emissor não mais reconhece a FEC para a qual havia atribuído uma etiqueta.

- 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 Etiqueta-FEC.

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

3.7 Planos de Controle e Encaminhamento no LDP

É importante esclarecer o que significa Plano de Controle (*control plane*) e Plano de Encaminhamento (*forwarding plane*) no LDP. Referências a estes planos são comuns na literatura sobre o LDP.

O Plano de Controle no LDP, são as regras ou operações que o LDP executa. Por exemplo, enviar mensagens LDP entre LSRs para estabelecer um LSP, usando as informações do Plano de Encaminhamento.

O Plano de Encaminhamento diz respeito as mensagens que os LSRs trocam para construir suas tabelas de roteamento (*routing*) e encaminhamento (*forwarding*) de pacotes. Está relacionado mais especificamente ao encaminhamento dos pacotes através dos links físicos de cada LSR. Alguns autores separam o roteamento e encaminhamento de pacotes. Neste trabalho subentende-se roteamento e encaminhamento de pacotes como funções executadas pelo Plano de Encaminhamento do LDP.

São exemplo de protocolos de roteamento usados para construir as tabelas de roteamento/encaminhamento IP: OSPF, RIP, IS-IS, BGP, etc.

É importante não confundir o roteamento IP com as funções realizadas pelo LDP. O LDP utiliza as informações do roteamento IP para criar suas tabelas de etiquetas (*Label Information Base - LIB*) basicamente associando uma etiqueta de entrada e uma

etiqueta de saída a cada FEC (prefixo de endereços IP) extraída da tabela de roteamento IP do LSR.

Na primeira vez que um novo pacote é roteado em um domínio MPLS, o roteamento se dá através do protocolo IP tradicional. Após este primeiro roteamento o LDP toma conhecimento da nova FEC (prefixo de endereço IP) através das informações do plano de encaminhamento IP, associa etiquetas MPLS a ela e assim tem condições de estabelecer um LSP. O próximo pacote e outros subseqüêntes, relacionados a esta FEC, serão roteados através do MPLS.

3.8 CR-LDP (*Constrained-Based Routing Protocol*)

De forma a permitir a configuração de LSPs sujeitos a restrições, o protocolo LDP foi estendido e renomeado para CR-LDP *(Contraint-based Routed* LDP) [JAMOUSSI, 2002].

A partir da inserção de parâmetros de especificação de Engenharia de Tráfego e Qualidade de Serviços (QoS) nas mensagens LDP de requisição de etiquetas, o CR-LDP tornou possível a reserva de recursos em LSPs MPLS. Esses parâmetros permitem que sejam requisitados valores específicos de taxa de pico ou PDR *(Peak Data Rat*e), taxa média ou CDR *(Commited Data Rat*e), variação máxima do retardo (*frequenc*y), reserva de banda, dentre outros.

Outras novas funcionalidades adicionadas pelo CR-LDP são:

- roteamento explícito: permite estabelecer um LSP com roteamento explícito;
- preempção entre LSPs: permite atribuir prioridades aos LSPs quando de seu estabelecimento. Um LSP menos prioritário, mesmo já estabelecido, pode ser preemptado pelo fato de LSP mais prioritário estar sendo estabelecido.

Vale salientar que a facilidade de implementar a engenharia de tráfego no MPLS é atualmente um dos grandes focos de interesse em relação ao uso do MPLS sobre as rede IP, com a finalidade de propiciar uma operação mais eficiente destas redes. Nesse contexto além do CR-LDP existem também extensões ao protocolo RSVP com

finalidades semelhantes às do CR-LDP, no que diz respeito a engenharia de tráfego, QoS e outros.

3.9 Conclusões do Capítulo

Este capítulo apresentou as características básicas e o funcionamento do protocolo LDP. Foi mostrado como ocorre o processo de troca de mensagens, as fases de estabelecimento, manutenção e encerramento das sessões LDP e as operações de gerência com etiquetas MPLS (*labels*). Foram definidos os significados de plano de controle e encaminhamento no LDP e foi apresentada uma visão geral do protocolo CR-LDP, que é uma extensão do LDP e trata do roteamento explícito de LSPs.

4 CONSIDERAÇÕES SOBRE A SEGURANÇA DO LDP

Este capítulo identifica ameaças as quais o LDP está vulnerável e discute maneiras através das quais essas vulnerabilidades podem ser amenizadas, conforme discutido na RFC 3036 [ANDERSSON, 2001].

4.1 Ataques Baseados em *Spoofing*

Para evitar ataques de *spoofing*[5] de etiquetas MPLS, é necessário assegurar que pacotes etiquetados tenham sido etiquetados por LSRs confiáveis e que as etiquetas inseridas nos pacotes são corretamente aprendidas pelos demais LSRs, que por sua vez geram e distribuem novas etiquetas baseadas nas recebidas.

Existem dois tipos de comunicações LDP que podem ser alvos de ataques do tipo *spoofing*.

4.1.1 Troca de Mensagens de Descoberta Transportadas via UDP

LSRs diretamente conectados no nível de enlace trocam mensagens LDP de Descoberta (mecanismo de Descoberta Básico do LDP) pelo link. A ameaça de *spoofing* de HELLOS Básicos pode ser reduzido por:

- aceitar HELLOS Básicos apenas nas interfaces as quais LSRs que são considerados confiáveis estão diretamente conectados.
- Ignorar HELLOS Básicos não endereçados ao endereço de grupo *multicast* da rede.

LSRs não diretamente conectados por enlace devem usar o mecanismo de descoberta estendido do LDP (*targetted hellos*) para indicar a intenção de estabelecer uma sessão LDP com um LSR específico. O LSR pode reduzir a ameaça de *spoofing* de

[5] Spoofing é uma técnica de falsificação do endereço de origem.

HELLOS Estendidos através de filtragem de pacotes, onde somente uma lista de acesso definida possui permissão de enviar este tipo de pacote.

4.1.2 Sessão LDP Transportada via TCP

O LDP especifica o uso da autenticação TCP baseada em MD5 para prover autenticação e integridade para as mensagens de sessão sobre o TCP.

A RFC 2385 [HEFERNAN, 1998] alerta que a autenticação TCP baseada em MD5 é atualmente considerada, por alguns, como insuficiente para ser aplicada ao LDP. Também salienta que uma opção baseada em TCP, similar, com um algoritmo de *hash* mais forte (cita o SHA-1 como exemplo) pode ser desenvolvido como uma alternativa. Porém até o momento nenhuma opção semelhante foi definida e/ou implementada. Vale salientar que no LDP, entre LSRs adjacentes, pode ser usado qualquer tipo de mecanismo de autenticação baseado em conexão de transporte TCP.

4.2 Confidencialidade

O LDP não fornece um mecanismo para prover confidencialidade na distribuição de etiquetas. Os requisitos de segurança de protocolos de distribuição de etiquetas são essencialmente iguais aos dos protocolos de distribuição de rotas IP. Provendo um mecanismo que assegure a autenticação e a integridade de suas mensagens, o LDP prove um nível de segurança que é tão bom quanto dos protocolos de distribuição de rotas como o BGP por exemplo, que inclusive adota o mesmo mecanismo para autenticação. Uma discussão mais aprofundada a respeito da necessidade de confidencialidade em protocolos de roteamento não está no escopo deste trabalho.

4.3 Negação de Serviço (*Denial of Service*)

O LDP possui dois alvos potenciais para ataques de negação de serviço.

4.3.1 Porta UDP 646 usada pelo Mecanismo de Descoberta do LDP

O administrador de um LSR pode tratar a ameaça de ataques de negação de serviço via Hellos do tipo básico assegurando que o LSR esteja diretamente conectado somente a outros LSRs confiáveis.

Interfaces com parceiros no interior do domínio MPLS não representam uma ameaça pelo fato de estarem no mesmo domínio, ou sob a mesma administração. Interfaces com parceiros externos ao domínio MPLS representam uma ameaça em potencial. O administrador pode reduzir esta ameaça conectando esses LSRs somente com pares externos confiáveis.

Ataques de negação de serviço via HELLOS Estendidos (mecanismo de descoberta estendido do LDP) são potencialmente uma ameaça mais séria. Essa ameaça pode ser tratada através da filtragem de HELLOS Estendidos usando listas de acesso que definem os endereços dos LSRs com os quais este tipo de comunicação é permitido. Porém, por outro lado, executar filtragem de pacotes consome recursos dos LSRs.

Em um ambiente onde um domínio MPLS confiável pode ser identificado, os LSRs das bordas podem ser usados para proteger os LSRs do interior desta nuvem contra ataques via HELLOS Estendidos. Isto pode ser alcançado através da filtragem destes HELLOS originados de fora do domínio MPLS confiável, aceitando somente aqueles endereços permitidos segundo uma lista de acesso definida. Esta filtragem protege os LSRs no interior do domínio MPLS porém consome recursos dos LSRs das bordas.

4.3.2 Estabelecimento de Sessões LDP na Porta TCP 646

Como outros protocolos que usam TCP, o LDP pode ser alvo de ataques de negação de serviço (DoS), como ataques baseados em SYN (componente da sinalização do protocolo TCP). O LDP não é mais e nem menos vulnerável a estes tipos de ataques que outros protocolos que usam TCP.

A ameaça deste tipo de ataques pode ser amenizada através das seguintes ações:

- Um LSR deve evitar escutar pelo estabelecimento de sessões LDP de forma promíscua. Ele deve somente escutar conexões específicas de pares já descobertos. Isso habilita aos mesmos, negar pacotes de ataques antes do seu processamento.
- O uso da autenticação TCP baseada em MD5 é valido. Porém o LSR receptor do segmento TCP deve calcular o *checksum* MD5 deste segmento antes de poder decidir se vai aceitar ou descartar o segmento SYN.
- O uso de mecanismos baseados em lista de acessos aplicados nas bordas da nuvem MPLS de maneira similar a sugerida acima para HELLOS Estendidos pode proteger os LSRs do interior da nuvem MPLS contra ataques originados de fora da nuvem.

4.4 Considerações sobre Autenticação e Integridade no LDP

O LDP conforme descrito na RFC 3036 [ANDERSSON, 2001] é um protocolo concebido pelo MPLS para a distribuição de etiquetas. Ele não define mecanismos próprios, num escopo por pacote, para autenticação, integridade e proteção anti-replay. Em contrapartida, como é orientado a conexão, ou seja, roda sobre o TCP, permite utilizar a autenticação TCP baseada em MD5 (RFC2385) [HEFFERNAN, 1998] também usada pelo BGP para prover autenticação e integridade em comunicações entre LSRs adjacentes.

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.

4.5 Requisitos de Segurança para o LDP

O protocolo LDP é usado para transmitir informações de roteamento MPLS que são captadas das tabelas de roteamento IP nos LSRs de Ingresso e introduzidas dentro

dos domínios MPLS. Dessa forma ambos os pacotes de sessão e de dados são vulneráveis a ataques que incluem:

- um adversário pode tentar modificar os pacotes (de sessão ou de dados).
- um adversário pode tentar roubar uma conexão LDP.
- um adversário 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 adversário pode tentar corromper a negociação entre dois pares LDP, de forma a ganhar acesso a chave compartilhada.

A confidencialidade não está sendo listada como um requisito baseado no fato que as informações transportadas nas mensagens LDP são informações de roteamento MPLS (etiquetas) e as mesmas não são de natureza confidencial. Podemos comparar as informações transportadas nas etiquetas MPLS às informações de roteamento de protocolos de distribuição de rotas IP como BGP e OSPF.

Da especificação atual do protocolo LDP (RFC 3036), o mecanismo de autenticação TCP baseado em MD5 definido, não atende todos os requisitos de segurança listados. Ele provê autenticação mútua entre o par local e o par adjacente quanto a origem da conexão, provê segurança para o tráfego de dados e sessão TCP em um escopo por pacote, mas não provê, num escopo por pacote, autenticação, integridade e proteção *anti-replay*, assim deixa a sessão LDP vulnerável a ataques.

Além das questões relacionadas à segurança o uso do mecanismo de autenticação TCP baseado em MD5 é aplicável apenas em comunicações LDP entre LSRs adjacentes, pois depende de uma conexão TCP estabelecida, sendo assim não pode ser aplicado em comunicações LDP entre LSRs não-adjacentes.

4.6 Trabalhos Correlatos

Nesta seção serão apresentados alguns trabalhos correlatos ao assunto tratado neste trabalho.

a) "**End to End Authentication for LDP** " [DE CLERCQ, 2001].

Este documento descreve uma proposta de autenticação fim a fim para o protocolo LDP e foi sugerido em forma de *draft* para o IETF.

São sugeridos dois métodos para viabilizar autenticação mútua entre LSRs não-adjacentes durante o estabelecimento de LSPs dentro do LDP. Ambos os métodos fornecem autenticação, integridade e controle contra ataques de repetição às mensagens LDP.

O primeiro método sugere o uso de certificação digital, onde um certificado seria anexado a cada mensagem LDP enviada. Os autores não deixaram claro como este ambiente seria implementado dentro do LDP, apenas fornecem uma visão geral de como este método poderia ser empregado, definem os TLVs e comentam os procedimentos necessários para a operação. Faltaram muitos detalhes importantes referentes a este método. Os autores não deixaram claro como pretendem implementar isso.

O segundo método proposto sugere um mecanismo de *Message Authentication Code* (MAC) anexado a cada mensagem LDP. Este mecanismo possibilita ao LER receptor autenticar as mensagens LDP recebidas, porém não aos LSRs intermediários. O método utiliza uma chave compartilhada entre o LSR de Ingresso e o LSR de Egresso, a qual também serve como chave secreta para um algoritmo MAC aplicado sobre as mensagens trocadas entre os LSRS.

Os autores não fizeram experimentos práticos, como implementação ou simulação das soluções sugeridas.

Através de uma análise aprofundada da proposta deste trabalho correlato, conclui-se que a proposta apresentava um erro arquitetural, fato reconhecido pelos seus autores através de contato por e-mail. A proposta foi projetada para autenticar fim a fim, o estabelecimento de LSPs dentro do LDP, porém ela não se aplica a maioria dos casos pois considera que ao solicitar um LSP para uma determinada FEC, o LSR de origem sabe qual é o LSR de destino que vai atender esta requisição. Isso não representa uma verdade na forma padrão de fucionamento do protocolo LDP.

Na forma padrão de funcionamento do LDP, o LSR de origem, durante o estabelecimento de um LSP, apenas tem conhecimento da FEC e deseja requisitar ou atribuir uma etiqueta (*label*) para esta FEC. Porém ele não tem conhecimento de qual LSR será a extremidade oposta para aquele LSP. A proposta deste trabalho correlato considera que o LSR sabe essa informação, e isto está incoerente.

Existem casos específicos ao qual a proposta deste trabalho correlato poderia ser aplicável, mas obviamente o escopo de aplicação fica drasticamente reduzido comparado com a pretensão inicial. Por exemplo, na aplicação descrita por [ROSEN, 2002] que trata de VPNs BGP/MPLS, é requerido aos LERs (LSRs de Ingresso e Egresso) do LSP incluir seus endereços IP de *host* (prefixo /32) no roteador central responsável pelo roteamento intradomínio (IGP – *Interior Gateway Protocol*). Assim o LSR origem tem condições de saber quem será o LSR da outra extremidade do LSP. Outras aplicações MPLS que utilizam condições semelhantes, nas quais a proposta deste trabalho correlato poderia ser aplicada: *Layer-3 VPNs*, *Layer-2 VPNs*, Hierarquias de domínio MPLS (*MPLS domain hierarchies*) e redes em centrais (*backbones*) sem uso de BGP.

b) "**Security Standarts for the Global Information Grid**" [BUDA, 2001].

No artigo, GIG (*Global Information Grid)* é definido como um conjunto global, interconectado fim a fim, de capacidades, processos associados e procedimentos para coletar, processar, armazenar, disseminar e gerenciar informações, sob demanda, das redes de computadores

O artigo apresenta uma visão geral dos requisitos para melhorar a infra-estrutura de segurança da GIG. O contexto para tornar mais rígida esta infra-estrutura global é desenvolver padrões comerciais que encorajem os produtos de TI a suportar a GIG. Serviços de infra-estrutura candidatos a incorporar uma maior rigidez na segurança são: serviços de sinalização, roteamento, gerenciamento, nomes e localização. Os padrões comerciais de interesse da GIG incluem tecnologias em diferentes estágios de maturidade, como ATM (*Assyncronous Transfer Mode*), MPLS (*Multi-protocol Label Switching*) e redes ópticas. Apenas estas três tecnologias são citadas.

As redes dentro da GIG são formadas por uma grande variedade de tecnologias de roteamento, *switching* (chaveamento de circuitos e/ou pacotes) e sistemas de transporte, dessa forma o artigo decompõe o tráfego de todas essas redes em três concepções básicas: usuário, controle e gerenciamento das comunicações, as quais são descritas no do documento.

O artigo também descreve as atividades do DOD (*Departament of Defense*) no sentido definir requisitos e padrões de segurança mais rígidos para roteadores e *switches* que implementam essas tecnologias de interesse da GIG.

Quanto aos protocolos definidos para distribuição de etiquetas dentro do MPLS, LDP (*Label Distribution Protocol*) e RSVP (*Resource Reservation Protocol*), ambos definem o uso da autenticação TCP baseada no algoritmo MD5 [HEFERNAN, 1998] para fornecer autenticação e integridade às mensagens de sinalização em contextos *hop by hop,* ou seja, entre LSRs adjacentes. Esta é uma solução que se adapta bem, porém esta longe de ser o ideal.

Em primeiro lugar, algumas aplicações requerem confidencialidade das mensagens de sinalização para proteger as atividades dos usuários, reduzir sua vulnerabilidade a análise de tráfego ou esconder informações sobre suas configurações de rede. Assim autenticação, integridade das mensagens, detecção de reenvio (*replay*) e confidencialidade devem estar disponíveis como serviços de segurança para que o usuário possa escolher.

Em segundo lugar, um sistema de autenticação rígido deve prover autenticação nos dois sentidos da comunicação e ser integrado a um sistema de gerenciamento de chaves. Um sistema de gerenciamento de chaves é com certeza muito mais complexo de ser implementando que uma simples proteção às mensagens do protocolo, adicionalmente é requerido um gerenciamento de chaves automatizado de modo a fornecer escalabilidade a solução. Tem sido usados argumentos de que um gerenciamento de chaves é muito complexo de ser implantado no ambiente, porém ignorar não é uma solução. Gerenciamento de chaves é por herança hierárquico. Chaves longas protegem chaves curtas. Protocolos fornecem mecanismos de defesa, como encaminhamento de segredos, que limitam o efeito do simples comprometimento de uma chave.

Em terceiro lugar, uma grande variedade de algoritmos de criptografia e tamanhos de chaves devem ser suportados para suportar diferentes aplicações, jurisdições, categorias de informações e configurações de rede. Estes devem incluir métodos populares como também formas de especificar algoritmos proprietários. O protocolo (LDP ou RSVP) deve acomodar estas escolhas para suportar negociações seguras de mecanismos e serviços. E por final, deve ser dada atenção a capacidades auxiliares como reforçar políticas de segurança, reportar erros e atualizar chaves de sessão dinamicamente.

c) "**Secure Border Gateway Protocol (S-BGP)"** [KENT, 2000].

O BGP (*Border Gateway Protocol*) o qual é usado para distribuição de rotas entre ASes (*Autonomous Systems*) é um componente crítico na infra-estrutura de roteamento da internet. Ele é altamente vulnerável a uma variedade de ataques maliciosos relacionados a verificação da autenticidade e legitimidade do tráfego de controle do BGP. Este artigo descreve uma arquitetura (S-BGP) segura e escalável para um sistema de autorização e autenticação que endereça a maioria dos problemas de segurança associados ao BGP. O artigo discute as vulnerabilidades e requisitos de segurança associados ao BGP, descreve a solução proposta pela arquitetura S-BGP, explica com ela endereça as vulnerabilidades e requisitos de segurança apontados e analisa as implicações de performance e operação da solução. Adicionalmente, o artigo realiza uma comparação da arquitetura S-BGP com outras soluções de segurança propostas ao BGP, que inclui a solução de autenticação TCP baseada em MD5, proposta por [HEFFERAN, 1998], também definida para o protocolo LDP na RFC 3036.

A solução (S-BGP) proposta para fornecer segurança à distribuição de rotas no BGP envolve um novo atributo de rota (Path) contendo "atestados", o uso do IPSec (*Internet Protocol Security*) e 2 Infra-estruturas de chave pública (PKIs) (tecnicamente falando, são empregadas duas hierarquias de certificação, porém elas possuem procedimentos comuns e etc., por isso podem ser consideradas como uma única PKI). Estes componentes são usados pelos BGP *Speakers* (roteadores com suporte a BGP que divulgam rotas BGP) para validar a autenticidade e a integridade de dados das rotas BGP recebidas e para verificar a identidade e autorização dos anunciantes destas rotas

BGP. As duas PKIs definidas pela solução são: uma PKI para alocação de endereços, onde cada certificado contém uma extensão privada (*private extension*) que especifica o conjunto de blocos de endereços que foram alocados para esta organização, e uma PKI para atribuir associações entre ASes (*Autonomous Systems*) e roteadores.

Quanto aos "atestados", são utilizados duas classes de "atestados", representados por um formato comum que são: atestados de endereços e atestados de rotas. Se uma organização possui mais de um AS terá atestados separados para cada AS.

Para a validação das atualizações de rotas (*route updates*) recebidas pelos BGP *Speakers*, são usados certificados e atestados, para verificar se o primeiro AS da rota foi autorizado a anunciar este bloco de endereços pelos donos do bloco de endereços, e que cada AS subsequente foi autorizado para anunciar a rota para o bloco de endereços pelos AS precedentes na rota.

A distribuição dos Certificados, CRLs (*Certificate Revogation List*), atestados de endereço e atestados de rotas são um problema dentro da solução. Os autores propõem o uso de dois níveis de repositórios (que poderiam ser acessados via LDAP – *Lightweight Directory Access Protoco*l) de onde qualquer roteador BGP poderia realizar o *download* de toda a base de certificados, CRLs e atestados.

A solução faz uso do IPSec da seguinte maneira: o protocolo ESP (*Encapsulation Security Payload*) (com criptografia NULA) é usado para prover autenticação, integridade e controle anti-replay num escopo ponto a ponto, ou seja, entre os BGP *Speakers*. O protocolo IKE (*Internet Key Exchange*) é usado para o gerenciamento de chaves em suporte ao protocolo ESP. A PKI proposta na solução para autenticação de rotas e ASes fornece os certificados necessários.

O artigo também faz considerações sobre consumo de processamento nos roteadores BGP, largura de banda consumida nas operações e consumo de espaço de armazenamento e memória.

d) "**Supporting Virtual Private Dial-Up Services Over Label Switching Based Networks**" [WU, 2000].

Para as emergentes VPNs (Virtual Private Networks) e o constante crescimento dos usuários móveis na internet, o uso de VPNs sobre arquiteturas de chaveamento baseado em etiquetas (*label switching architectures*), como o MPLS (*Multiprotocol Label Switching*), apresentam grandes vantagens, devido ao seu suporte nativo a VPNs.

Tendo em vista que atualmente o MPLS é uma das tecnologias mais promissoras para suportar a infra-estrutura de backbone da internet global, muitos estudos tem sido realizados no sentido de prover VPNs sobre MPLS. A maioria eles focalizam a integração de diferentes *sites* de uma VPN sobre o MPLS. Em adição ao problema da integração, prover mobilidade aos usuários VPN é também um ponto crucial.

Os protocolos utilizados atualmente para prover VPNs sobre a Internet, PPTP e L2TP, inserem uma grande quantidade de controle aos cabeçalhos dos pacotes, o que gera um *overhead* significativo nestas comunicações. Como diferentes semânticas podem ser inseridas nos *labels* (etiquetas) de redes como o MPLS, a informação relacionada às VPNs pode ser facilmente anexada a estes *labels*.

Neste artigo, um novo mecanismo para fornecer mobilidade aos usuários de VPNs é proposto. Anexando informações das VPNs às etiquetas (*labels)* MPLS, de maneira a formar um túnel LST (*Label Switching Tunnel*), o protocolo sendo proposto LSTP (*Label Switching Tunnel Protocol*) reduz o *overhead* das VPNs significativamente.

O artigo apresenta uma visão geral das tecnologias VPN atualmente empregadas e após apresenta os detalhes do protocolo LSTP sendo proposto. Um dos requisitos básicos para o sucesso do ambiente é que a Internet tenha suporte a tecnologia MPLS a nível de *backbone*.

A comunicação baseia-se em cliente servidor. No lado servidor (ou seja, na rede privada) temos o AS (Servidor de autenticação no *site* VPN) e o ER (roteador MPLS de egresso, que faz a ponte entre o *backbon*e MPLS e a rede local). No lado cliente temos o MR (usuário móvel da VPN) e o IRS (Servidor de Acesso Remoto do Provedor de Serviços Internet do MR).

Basicamente, o MR através de acesso discado liga para o seu ISP (Provedor de Serviços Internet), é atendido pelo IRS que autentica suas credenciais e estabelece um túnel VPN tradicional (PPTP ou L2TP) até o MR. Numa segunda etapa, o IRS através do protocolo LSTP sugerido estabelece um túnel MPLS (LST) bidirecional como o ER através do *backbone* MPLS (na visão do MPLS são criados dois LSPs (*Label Switch Path*s), um para cada direção da comunicação). As informações de autenticação do MR e etiquetas MPLS (*labels*) gerados pelo IRS são repassados ao ER através do protocolo LDP (*Label Distribution Protocol*) do MPLS no momento do estabelecimento dos LSPs entre o IRS e o ER. Para este propósito é utilizado o campo "*options*" das mensagens LDP (por ex. em uma mensagem LDP Label Mapping). Com as informações do MR, recebidas através do LDP, o ER tem condições de cadastrar as etiquetas MPLS em sua tabela local, de modo a criar o LSP solicitado, e repassar a informação de autenticação do MR ao AS, que vai validar as credenciais do MR. Após a etapa de autenticação os túneis são efetivamente estabelecidos e a tráfego do MR é escoado através destes.

Este trabalho foi citado como correlato, pois se baseia na confiabilidade do protocolo LDP para estabelecer túneis (LSPs) entre LSRs não-adjacentes.

4.7 Conclusões do Capítulo

Este capítulo apresentou algumas ameaças as quais o LDP está vulnerável e discutiu maneiras de amenizar estas vulnerabilidades. Por final foram apresentados alguns trabalhos correlatos relacionados a segurança do protocolo LDP.

5 AUTENTICAÇÃO FIM A FIM PARA O LDP

Este capítulo apresenta a proposta de autenticação fim a fim para o LDP.

5.1 Estabelecimento de um LSP (*Label Switching Path*) entre LSRs (*Label Switching Routers*) Não-adjacentes

Esta seção explica como o protocolo LDP opera para estabelecer um LSP entre dois LSRs não-adjacentes LERA e LERB (Figura 11).

Nesta rede MPLS LerA e LerB são respectivamente, ingresso e egresso do LSP e são LSRs não-adjacentes entre si. O Lsr1 e Lsr2, Lsr2 e Lsr3, LerA e Lsr1, Lsr3 e LerB, são LSRs adjacentes.

Considera-se que:

a) o LDP está configurado para operar no Modo de Distribuição Sob Demanda (isso significa que um LSR apenas vai distribuir uma etiqueta 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.

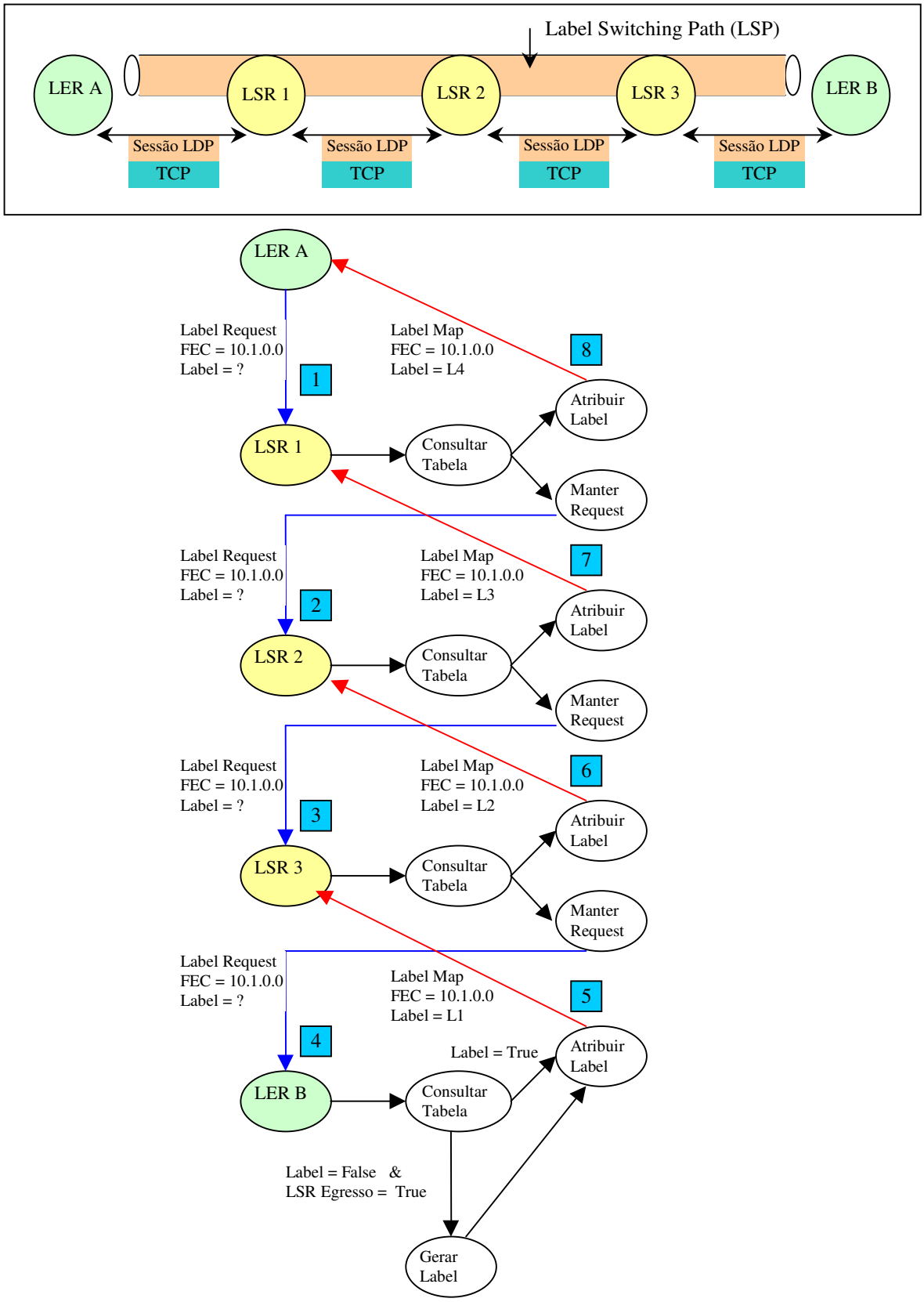


Figura 11. Passos executados pelo LDP para estabelecer um LSP entre os LSRs não-adjacentes LERA e LERB.

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 etiquetas MPLS.

Após o estabelecimento das sessões LDP entre os LSRs adjacentes temos esta visão do plano de controle do LDP:

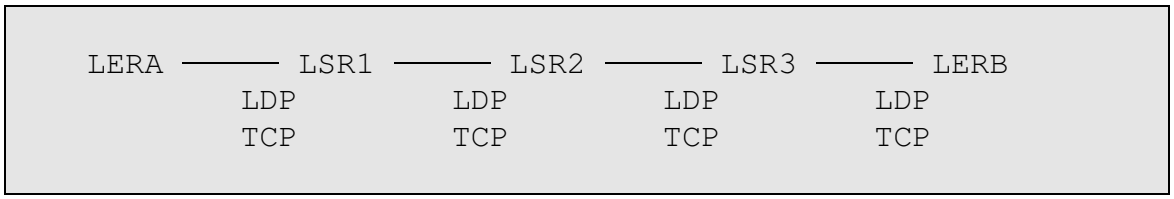


Figura 12. Visão do plano de controle do LDP após o estabelecimento das sessões TCP/LDP (fase ativa do LDP)

Agora que as sessões LDP estão 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.

Então:

a) No passo 1, o LER envia uma mensagem LDP LABEL REQUEST para o LSR1 solicitando uma etiqueta (*label*) para a FEC 10.1.0.0/8;

b) No passo 2, O LSR1 realiza uma procura em sua tabela de etiquetas local (*Label Information Base - LIB*) e percebe que não possui uma etiqueta 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 uma etiqueta 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 uma etiqueta L1 para a FEC 10.1.0.0/8 e envia uma mensagem LDP LABEL MAPPING para o LSR3, informando esta etiqueta.

d) No passo 6, o LSR3 insere a etiqueta recebida 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 a etiqueta L2 para o LSR2 através de uma mensagem LDP LABEL MAPPING.

e) No passo 7, O LSR2 insere a etiqueta em sua LIB e percebe que possui uma requisição pendente para a FEC 10.1.0.0/8 originada por LSR1. Então gera e envia uma etiqueta 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 (etiqueta L4) no passo 8.

f) Quando o LERA recebe a mensagem LDP LABEL MAPPING com a FEC 10.1.0.0/8 do LSR1 (via a sessão TCP/LDP entre o LERA e o LSR1), ele insere a etiqueta recebida 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 seguinte visão da pilha de protocolo:

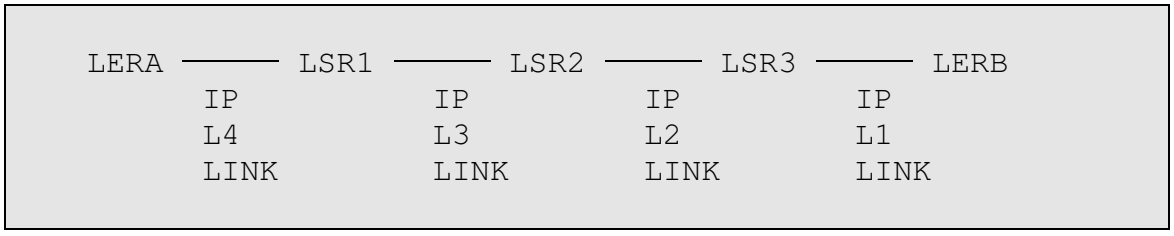


Figura 13. Visão do Plano de Encaminhamento do LDP após o estabelecimento do LSP entre LERA e o LERB

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 a etiqueta L1 em relação ao seu par adjacente LSR3 (Figura 13), 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 e outros.

5.2 Proposta de Autenticação Fim a Fim para o LDP

5.2.1 Introdução

A proposta deste trabalho é prover 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.

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 SSL,TLS e outras não se aplicam ao caso. Para tratar esta situação, a solução proposta neste trabalho 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.

A solução proposta faz uso de um mecanismo de autenticação, baseado em criptografia assimétrica (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*[6] 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

[6] Nonce: são técnicas utilizadas em protocolos de rede para detectar o reenvio de mensagens já recebidas em uma comunicação anterior, de forma a proteger contra ataques de repetição [ABADI, 1996].

solução foi projetada para ser utilizada com IP versão 4 (ipv4) e pode ser aplicada ao protocolo CR-LDP sem necessidade de adaptações.

5.2.2 Modelo de Autenticação Proposto

Foram definidos dois novos Tlvs ao LDP para prover a autenticação, "Tlv de Nonce" e "Tlv de Hash". As mensagens LDP envolvidas no processo de autenticação são: LABEL MAPPING, LABEL REQUEST e LDP NOTIFICATION. Estes três tipos de mensagens fornecem condições ao LDP de solicitar e atribuir etiquetas para o estabelecimento de LSPs e notificar falhas referentes a estas operações.

Como um cenário de exemplo (Figura 14), 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 (*Label Request*) enviada pelo LERA e retorna uma mensagem de mapeamento LDP (*Label Mapping*) autenticada ao LERA.

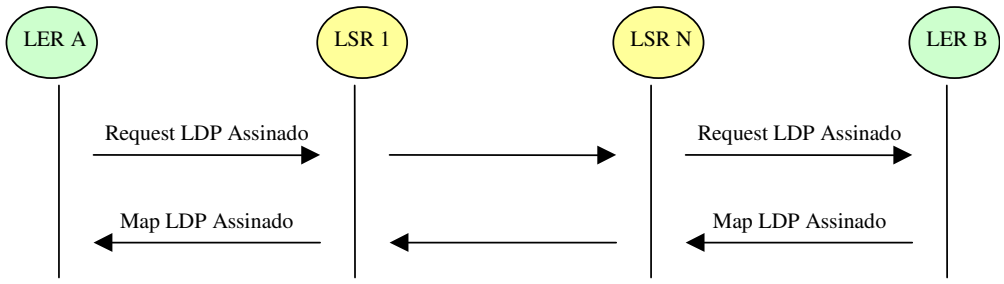


Figura 14. Diagrama da autenticação fim a fim proposta para o LDP.

Para solicitar o 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*[7]):
- baseado no conteúdo desta mensagem gera um resumo da mensagem aplicando um algoritmo de resumo de mensagens (*hash*);
- o valor *hash* é cifrado com a chave privada do LERA e inserido na mensagem LDP, juntamente com o identificador de LSR (LSR-ID) do LERA;
- o LERA envia a mensagem LDP ao próximo LSR no caminho (next hop) em direção ao LERB.

Os LSRs do caminho (intermediários) que não podem atender a requisição solicitada repassam os campos da autenticação transparentemente até que chegue ao 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. (As chaves públicas dos LSRs autorizados são informadas manualmente na configuração de cada LSR);
- decifra o valor *hash* recebido na mensagem de requisição utilizando a chave pública do LERA, assim verifica a autenticidade do LERA e pode confiar que a mensagem foi gerada por este LSR;
- da mesma forma que o LERA, o LERB gera um resumo (*hash*) da mensagem recebida 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 (por exemplo pode verificar se o tempo de transmissão da mensagem

[7] *Timestamp*: Identificação única baseada em tempo (data/ hora) com objetivo de protocolar um evento específico.

ultrapassou um limite de tempo definido), assim mantém um controle contra ataques de repetição.

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 uma etiqueta 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 cifra 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 baseado no resultado da autenticação executa ou não o estabelecimento do LSP com o LERB.

5.2.3 Novos TLVs e Tipos Definidos ao LDP pela Solução de Autenticação

Foram definidos novos TLVs ao LDP para concretizar a solução de autenticação: TLV de Hash e TLV de Nonce, e também um novo código de Status ao TLV de Status.

5.2.3.1 TLV de Hash

Foi definido um novo TLV para transportar um valor *hash* cifrado.

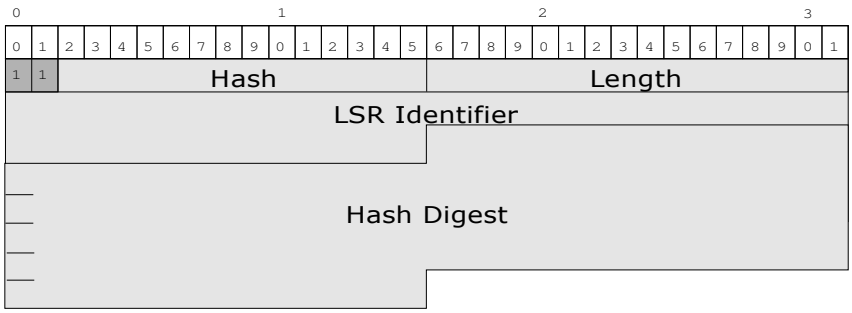


Figura 15. TLV de Hash

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 etiquetas (*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 5.2.4.1. O TLV de HASH deve obrigatoriamente ser anexado ao final da mensagem LDP, como o último TLV da mensagem. Este valor *hash* será cifrado 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 5.4. Neste caso a função hash gera uma saída e 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".

5.2.3.2 TLV de Nonce

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

Figura 16. TLV de Nonce

U-bit e F-bit seguem a mesma descrição anterior (item 5.2.3.1).

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.

5.2.3.3 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.

5.2.3.4 Considerações sobre o Registro dos TLVs Definidos pela Solução

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.

5.2.4 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.

5.2.4.1 Procedimentos ao ENVIAR 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 etiquetas (*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:

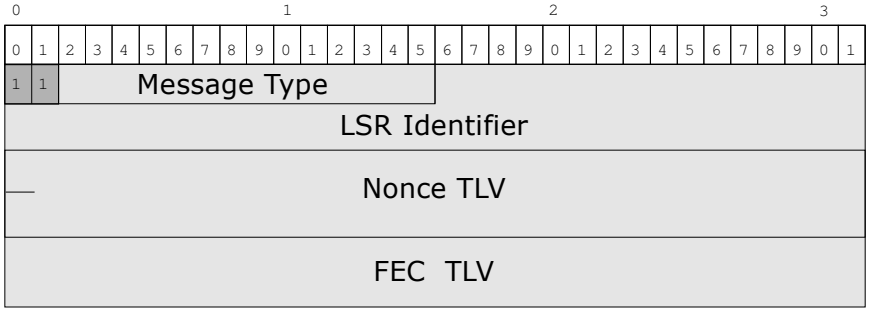


Figura 17. Entradas para mensagens LDP LABEL REQUEST e LABEL MAPPING

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

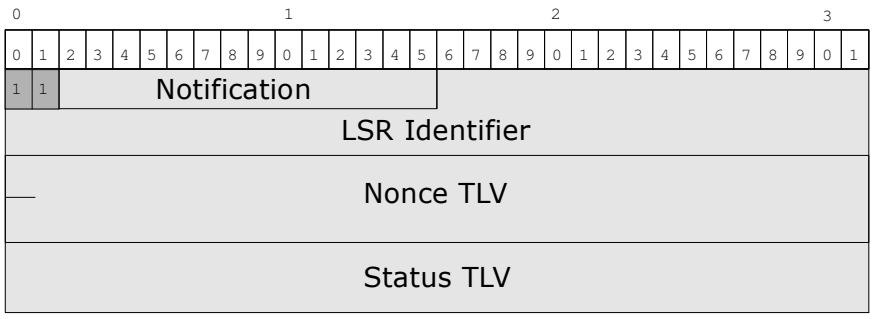


Figura 18. Entrada para mensagens LDP NOTIFICATION

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

5.2.4.2 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 autenticação. Se a operação for realizada com sucesso significa que o remetente é autêntico;

d) gerar um valor Hash da mensagem recebida. As entradas de dados devem ser formados da mesma maneira conforme foi descrito na seção 5.2.4.1. 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.

5.3 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 etiquetas 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.

Os algoritmos e tamanho de chaves sugeridos para as funções de *hash* e criptografia assimétrica serão discutidos na seção 5.4.

A seguir serão descritos os mecanismos adotados para prover a solução de autenticação:

a) Distribuição de Chaves

O método de autenticação proposto considera 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 a autenticação.

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, solução a qual será discutida na seção 5.5.

b) 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.

c) 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.

d) 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.

5.4 Considerações sobre Algoritmos de função Hash e Criptografia Assimétrica Sugeridos para a Solução

5.4.1 Algoritmos de *Hash*

Para função *Hash* sugere-se o algoritmo SHA-1 (*Secure Hash Algorithm*) [STALLINGS, 1999], com *digest* de 160 bits, ou seja, o algoritmo gera como saída uma string com 20 bytes de tamanho.

Para implementações futuras sugere-se uma análise do SHA-256 e SHA-512 [NIST, 2000].

5.4.2 Algoritmos de Criptografia Assimétrica

Para as funções de criptografia assimétrica sugere-se nesta proposta duas opções de algoritmos: Curvas Elípticas e RSA (Rivest-Shamir-Adleman). Literatura sobre ambos pode ser encontrada em [STALLINGS, 1999].

Considerando o uso do algoritmo SHA1, 160 bits para a função *hash* da autenticação, teremos uma entrada com 20 bytes de tamanho para a função de criptografia assimétrica.

Sugestão 1. Algoritmo de Curvas Elípticas:

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. Considerando que a entrada será uma string com 20 bytes de tamanho, o resultado da função de criptografia assimétrica aplicando o algoritmo de Curvas Elípticas será uma string com o mesmo tamanho da entrada, ou seja, 20 bytes.

Dessa forma sugere-se este algoritmo de criptografia assimétrica como o mais indicado para a solução de autenticação proposta neste trabalho.

Sugestão 2. Algoritmo RSA (Rivest-Shamir-Adleman):

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 cifrada 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 cifrada 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.

A solução de autenticação sugerida como a mais indicada por esta proposta é o algoritmo "SHA1, 160 bits" para a função *hash* e o algoritmo de "Curvas Elípticas" para a função de criptografia assimétrica.

5.5 Solução para 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.

Solução proposta:

a) eleger uma lista de Autoridades Certificadoras raiz (ACs) para emitir os certificados para a solução. Pode-se utilizar ACs independentes (Verisign, Certsign, etc.) ou soluções locais, por exemplo o software OPENCA (http://www.openca.org) ou outros, que envolvam menos custo.

b) todos os LSRs envolvidos no processo de autenticação, por exemplo os LSRs de um domínio MPLS, devem ter armazenado em sua configuração local o certificado digital das ACs raiz da lista eleita. Autoridades de Registro (AR) intermediárias, da árvore da AC raiz também podem emitir certificados, porém para isso todos LSRs envolvidos precisam manter em sua configuração local, também o certificado da AR intermediária corrente e de todas as ARs superiores a ela na árvore de AC raiz. Isso presume que no processo de implantação do LSR os certificados digitais das ACs raiz e ARs intermediárias devem ser localmente carregados. Por exemplo, o LSR pode se comunicar com cada AC raiz e ARs intermediárias e solicitar o seu certificado digital.

Sugere-se adotar poucos níveis de ARs emitindo certificados para a solução, dentro da árvore da AC raiz eleita, por causa da limitação de armazenamento dos LSRs (Flash ROM). Pelo mesmo motivo sugere-se não adotar uma lista extensa de ACs raiz.

c) todos os LSRs envolvidos que desejarem se autenticar, precisam ter um certificado emitido por alguma das ACs raiz ou por uma de suas intermediárias;

d) O prazo de validade padrão de um certificado digital de uma AC raiz (o qual é auto-assinado) é algo em torno de 10 anos, logo não seria necessário atualizar este certificado durante este período;

e) o emissor das mensagens LABEL MAPPING, LABEL REQUEST ou LDP NOTIFICATION deve enviar o seu certificado digital junto com a mensagem LDP. Isso implica que além dos certificados digitais das ACs raiz o próprio certificado digital do LSR também deve ficar armazenado em sua configuração local (*Flash ROM*). Também

durante o processo de implantação do LSR este deve solicitar um certificado digital para alguma das ACs raiz da lista ou intermediárias.

Para este propósito deverá ser definido um novo tipo de certificado "object.id", ou seja, uma extensão de certificado digital que terá a finalidade de autenticar fim a fim os extremos de um LSP dentro do LDP. Não serão abordados os detalhes da criação desta extensão de certificado digital neste trabalho, pois o objetivo é dar uma visão de como a certificação digital seria usada dentro do ambiente da proposta para distribuir as chaves públicas. Uma especificação detalhada da solução será sugerida como trabalhos futuros.

Deverá também ser definido um novo "TLV de Certificado" para o LDP, de forma a armazenar o certificado digital do originador da mensagem LDP. Por exemplo:

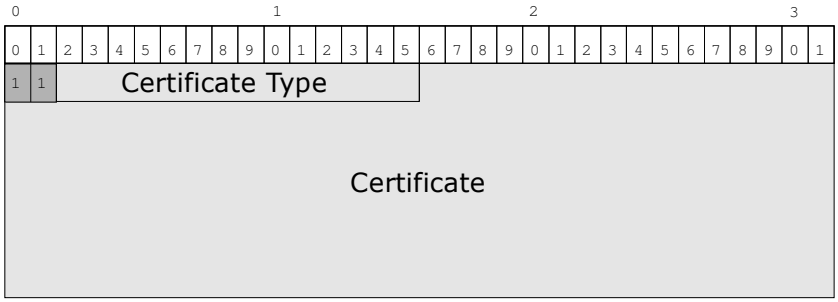


Figura 19. Exemplo de TLV de Certificado Digital

O campo "Certificate Type" serve para informar o tipo ou finalidade do certificado transportado. Um campo de 2 bytes de tamanho é suficiente. O campo "Certificate" serve para armazenar o certificado digital do remetente, seu tamanho depende tipo de certificado transportado, mas terá aproximadamente 1 Kbyte .

Pode-se adotar um *cache* de certificados em cada LSR, assim se um LSR remoto já passou o seu certificado em outra oportunidade, sendo que o mesmo ainda está no *cache*, ele não precisaria reenviar novamente. Porém, para este propósito precisa ser definido, todo um mecanismo de negociação ao LDP que viabilize que os LSRs que estão estabelecendo o novo LSP troquem informações de forma a saber se o remetente precisa ou não enviar o seu certificado digital. Porém este aspecto está fora do escopo desta versão da proposta e deverá ser avaliado em estudos futuros.

f) ao receber uma mensagem LDP cuja autenticação deve ser processada (caso o LSR receptor for egresso ou ingresso para FEC da mensagem), com posse dos certificados digitais da lista de ACs raiz e ARs intermediárias possíveis, os quais estão auto-assinados, o LSR consegue validar qualquer certificado emitido por elas. Considerando que todos os LSRs obrigatoriamente precisam ter certificados emitidos por alguma das ACs da lista ou por suas intermediárias, ele sempre terá condições de validar o certificado recebido do LSR originador da mensagem.

Dessa maneira o LSR receptor consegue validar/confiar no certificado enviado pelo originador da mensagem LDP e utiliza a chave pública informada neste certificado para verificar a assinatura do campo "Hash Digest" do TLV de Hash o qual foi cifrado com a chave privada do LSR originador.

Não necessariamente o LSR receptor precisa autorizar qualquer origem que possua um certificado emitido por alguma das ACs raiz ou intermediárias da lista. Pode-se adotar alguma forma de definir que somente LSRs específicos, dos que possuem certificados válidos emitidos pelas ACs da lista, tem acesso autorizado ou não autorizado em relação ao LSR receptor. O certificado digital da AC raiz e intermediárias, localmente armazenado, tem por objetivo validar se o certificado apresentado pelo LSRs é válido e confiável.

Um aspecto negativo desta solução de distribuição de chaves é que a mesma gera um acréscimo considerável de *overhead* ao protocolo LDP.

5.6 Conclusões do Capítulo

Este capítulo descreveu como ocorre o estabelecimento de um LSP entre LSRs não-adjacentes, de modo a situar o contexto de aplicação da solução proposta neste trabalho. A solução de autenticação fim a fim proposta para o LDP foi apresentada e discutida, e por final foi discutida uma solução alternativa para distribuição de chaves no ambiente da solução, baseada em certificação digital.

6 RESULTADOS DA IMPLEMENTAÇÃO

Este capítulo descreve a implementação da autenticação fim a fim proposta para o protocolo LDP.

6.1 Ferramentas Utilizadas

A plataforma Linux foi escolhida, pois além de ser um ambiente bastante difundido e amigável, possui a maior gama de projetos relacionados ao LDP cujo código fonte é aberto (*open source*). Todos os projetos de implementação do LDP analisados estão em um estágio de desenvolvimento incompleto [LEU, 2000], [AYAME Project, 1999].

Optou-se por adotar uma implementação de código fonte aberto, pois pela demanda de desenvolvimento necessária ficaria inviável a implementação de todo ou de uma parte relevante do protocolo LDP apenas para validar a autenticação proposta que é o escopo deste trabalho.

O projeto de código aberto selecionado, que implementa o LDP e MPLS na plataforma linux é denominado "*MPLS for Linux*" (http://sourceforge.net/projects/mpls-linux/), conforme [LEU, 2000] e está vinculado ao grupo "source forge" (http://sourceforge.net).

Este projeto está subdividido em dois módulos principais MPLS-LINUX e LDP-PORTABLE.

O objetivo do módulo MPLS-LINUX é implementar o MPLS no plano de encaminhamento (*forwarding*) do núcleo (*kernel*) do Linux. Este módulo possui uma interface de comandos que permite a criação de LSPs estaticamente, ou seja, por configuração explicita. Quanto a implementação o projeto MPLS-LINUX esta mais adiantado e mais estável que o projeto LDP-PORTABLE.

O módulo LDP-PORTABLE se utiliza das funções do MPLS-LINUX para criar LSPs (*Label Switched Paths*) dinamicamente. O Objetivo do módulo LDP-PORTABLE é servir como exemplo, uma tentativa inicial de implementar a especificação do protocolo LDP, definida pela RFC 3036, para a plataforma Linux e futuramente também para outras plataformas. A implementação do LDP-PORTABLE possui pendências de implementação e ainda está instável, porém para o propósito de servir de protótipo para a implementação e validação da proposta de autenticação deste trabalho, ele atende aos requisitos básicos.

As versões utilizadas na implementação do protótipo deste trabalho foram:

- LDP-PORTABLE, versão 0.200, disponibilizada em 02 de setembro 2002. Disponível por: http://prdownloads.sourceforge.net/mpls-linux/ldp-portable-0.200.tar.gz?download
- MPLS-LINUX, versão 1.170, disponibilizada em 02 de setembro 2002. Disponível por: http://prdownloads.sourceforge.net/mpls-linux/mpls-linux-1.170.tar.gz?download

No módulo LDP-PORTABLE foi inserido o código apropriado para implementar as funcionalidades de autenticação fim a fim proposta para o protocolo LDP neste trabalho.

O módulo LDP-PORTABLE, na versão utilizada está integrado à plataforma ZEBRA (http://www.zebra.org). O projeto ZEBRA 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 o seguintes protocolos de roteamento dinâmico: BGP, OSPF, IS-IS, RIP e atualmente protocolo LDP do MPLS.

6.2 Definições Adotadas na Implementação do Protótipo

Esta seção tem por objetivo especificar algoritmos, mecanismos de *nonce* e outras definições específicas que foram utilizadas na implementação do protótipo. Não

necessariamente foram utilizados os mesmos algoritmos e definições sugeridas como as mais apropriados para a solução, pois o protótipo esteve 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.

6.3 Descrição da 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 cifrado com a chave privada do LSR no envio e decifrada 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
} mplsLdpLblMapMsg_t;
```

O arquivo ldp_nortel.h também define os protótipos das funções utilizada no aruivo 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 forncer 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 nonceTlcExists 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 nonceTlcExists 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 nonceTlcExists 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 nonceTlcExists 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 nonceTlcExists 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 cifrado (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_HASHTLV_FIXLEN;
  hashTlv->lsrAddress = lsrAddress;
  hashTlv->labelSpace = labelSpace;
  for(i = 0; i < signLen; i++)
```

```
        hashTlv->hashDigest[i] = rsaSign[i];
  LDP_EXIT(NULL, "setupHashTlv");
  return MPLS_HASHTLV_FIXLEN + MPLS_TLVFIXLEN;
}
// 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_identifier++, us_attr, \
  g->lsr_identifier);
  //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_mesg * 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 etiquetas para seus pares LDP se estes solicitarem este procedimento, e justamente para realizar estas solicitações de etiquetas 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 autênticação 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
#ifndef 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 autênticação 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 autenticate
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/mplsd/mplsd.c e zebra/mplsd/Makefile:

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

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

```
//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 necesssárias para o funcionamento do ambiente:

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

m) alterações efetuadas no arquivo "zebra/mplsd/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_mplsd_OBJECTS = ... ldp_auth.$(OBJEXT)
LIBS = ... -lssl
// END:  AUTH
```

6.4 Instalação das Ferramentas no Linux

Esta seção tem por objetivo mostrar os passos executados para instalar as ferramentas utilizadas no ambiente de testes (Figura 20), na plataforma linux, com suporte ao MPLS/LDP habilitado.

É importante ficar claro que o LDP-PORTABLE é uma plataforma de código fonte aberto (*open source*) e ainda está em 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 pacote LDP-PORTABLE são:

- 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 mpl-linux (*patch)* ao kernel do linux de modo a fornece suporte a MPLS.

- 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 (passos detalhados podem ser encontrados no ANEXO 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.

6.5 Configuração e Execução do LDP no Ambiente de Testes

Foi configurado um ambiente de testes em duas máquinas Linux nomeadas como LERA e LERB respectivamente (Figura 20), para verificar o funcionamento da autenticação fim a fim implementada ao LDP.

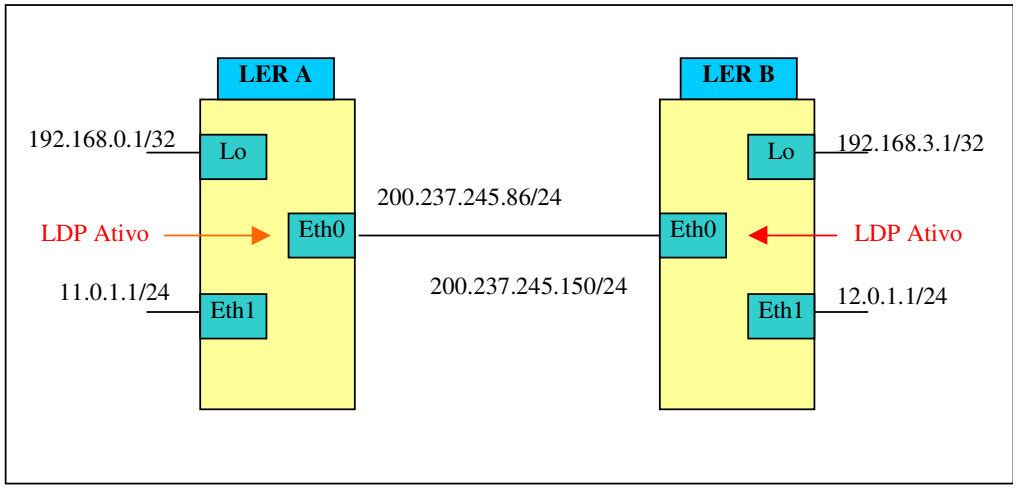


Figura 20. Ambiente de testes utilizado para verificar a autenticação implementada

A Figura 20 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 eqüivale 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:

```
LERA:
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

LERB:
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 etiquetas 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).

6.5.1 Visualização do Funcionamento do LDP no Ambiente de Testes

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 (Figura 20) como forma de exemplificar os resultados exibidos pelos comandos de vizualização 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
Propogate 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 neighbords
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 etiqueta(*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 viazualizar 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)
```

6.6 Resultados Obtidos

Esta seção tem por objetivo apresentar a análise dos resultados obtidos na fase de implementação do protótipo.

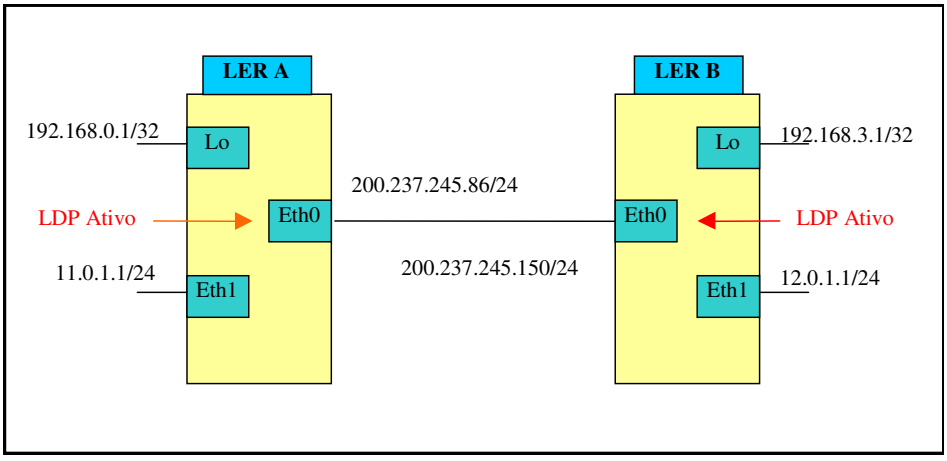
Na configuração *defaul*t, o LDP-PORTABLE inicializa o LDP no Modo de Controle Ordenado e Distribuição NÃO SOLICITADA. Se o LDP-PORTABLE, na versão utilizada neste trabalho (versão 0.200), for configurado para operar no Modo de Distribuição SOB DEMANDA a versão fica instável e a execução é encerrada 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 em uma máquina linux, 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.

Nesta seção, será retomado o ambiente de testes apresentado na Figura 20, o qual foi copiado para esta posição do texto de modo a permitir um melhor acompanhamento da interpretação desta seção.



Ambiente de testes utilizado para validar a proposta deste trabalho (cópia da Figura 20)

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 tornen-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 etiquetas 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 etiquetas 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.

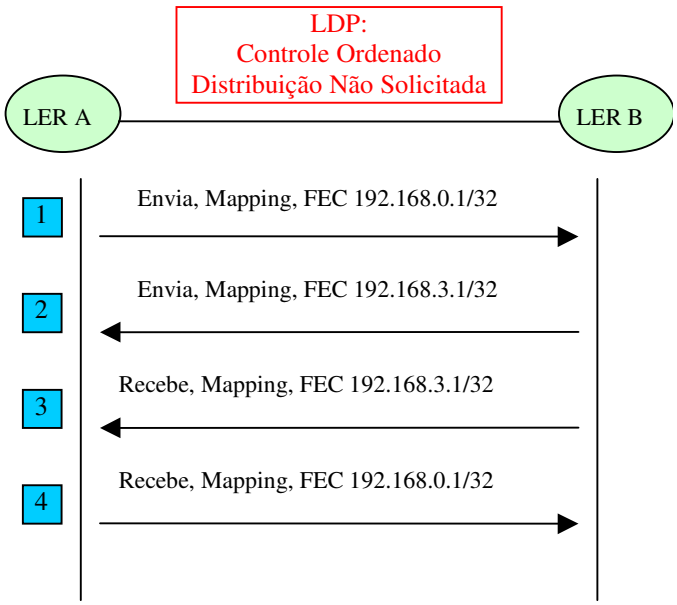


Figura 21. Diagrama de seqüência das mensagens de troca de *labels* ocorridas durante a execução do ambiente de testes.

O diagrama de seqüência apresentado na Figura 21 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 explicação de cada evento da Figura 21, ressalta em cada um dos arquivos de *log* (Figuras 22, 23, 24, 25, 26 e 27), com quadros de texto em negrito o processo de autenticação ocorrido.

A Figura 21 mostra a seqüência exata dos eventos ocorridos durante a execução do ambiente de testes (Figura 20). 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 LEBA 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 subseqüente).

```
--------------- Inicio do Bloco de LOG -----------------------------
...
// inicia a função que lê seqüêncialmente 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 o hash cifrado 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 = ¨Œ-™_ýÆ],™3½t_"¶□CÔy‰_
// assinatura RSA, com chave de 128 bits, aplicada sobre o valor hash
PRT: setupHashTlv: signLen = 128
PRT: setupHashTlv: sign =
ÊÊê§_GkGÎbéS3ƒÐã_2_Äœè<Ë,/N÷+B>ä°-1¿õâ__,Il ª‡g%8Niµ;²0íåjH-«²A
  'ò º£Bp<V¬Ò6Ê¼÷Ð®ñ¸²ÔI}úAð'Š"¤åªÖï
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 cifrado com a chave
// privada do LERB)
OUT:      hashTlv data:
OUT:      lsrAddress = c0a80301, labelSpace = 0, hashDigest =
ÊÊê§_GkGÎbéS3ƒÐã_2_Äœè<Ë,/N÷+B>äº–1¿õâ__,Ilª‡g%8Niµ;²0íåjH–«²A
  'òº£Bp<V¬Ò6Ê¼÷Ð®ñ¸²ÔI}úAð'Š"¤åªÖï
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 22. 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* subseqüente):

```
--------------- 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 1° 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 2° 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 cifrado
// com a chave privada do LERB
OUT:      hashTlv data:
OUT:      lsrAddress = c0a80301, labelSpace = 0, hashDigest =
ÊÊê§_GkGÎbéS3ƒÐã_2_Äœè<Ë,/N÷+B>ä°–1¿õâ__,Il ª‡g%8Niµ;²0íåjH–«²A
  'ò°£Bp<V¬Ò6Ê¼÷Ð®ñ¸²ÔI}úAð'Š"¤å ªÖï
OUT: LABEL MAPPING MSG ***END***
// tamanho da msg Label Mapping decodificada + tamanho do cabeçalho LDP
OUT: Mesg 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
_ldp_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 =
ÊÊê§_GkGÎbéS3ƒÐã_2_Äœè<Ë,/N÷+B>ä°–1¿õâ__,Il ª‡g%8Niµ;²0íåjH–«²A
  'ò°£Bp<V¬Ò6Ê¼÷Ð®ñ¸²ÔI}úAð'Š"¤å ªÖï
// 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 procesamento 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 23. 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* subseqüente):

```
--------------- Inicio do Bloco de LOG ----------------------------
...
// imprime mensagem confirmando o recebimento da mensagem Label Mapping do
// LERB em relação a FEC 192.168.3.1/32
OUT: Label Mapping Recv from c0a80301:0 for c0a80301/32
_ldp_global_add_outlabel
// instala na FIB o label recebido como label de saida 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 LERB foi modificada propositalmente
// no arquivo ldp_peers do LERA (causando uma falha de autenticação), pois
// o LERA não pode decifrar 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 mensagen 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 24. 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* subseqüente):

```
--------------- 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 o hash cifrado 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_ÈÒ¢ÞJoöŠ'™—¾³e¿„(™_;è„ëUÇ2_ð¸□¥e%éš□yÏ”wÉ“>E_”_"£å6æèPáEÿ_&½ÌyíÁTÔ
„SÖÉî¥hÖ@ùyÃ_ƒÃÈäX,&Pm□¼äü¬ñï…Œé*×å,>†é~SåÀYA_=%¶□.@©p
EXIT: setupHashTlv
// imprime campos e TLVs da mensagem Label Mapping que será enviada
OUT: LPD 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 cifrado com a chave
// privada do LERA)
OUT:      hashTlv data:
OUT:      lsrAddress = C0A80001, labelSpace = 0, hashDigest =
c_•E±_ó_ÈÒ¢ÞJoöŠ'™—¾³e¿„(™_;è„ëUÇ2_ð¸¥e%éš yÏ”wÉ“>E_”_"£å6æèPáEÿ_&½ÌyíÁTÔ
„SÖÉî¥hÖ@ùyÃ_ƒÃÈäX,&Pm¼äü¬ñï…Œé*×å,>†é~SåÀYA_=‰¶.@©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 25. 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 subseqüente):

```
--------------- Inicio 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_mesg
// 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: LPD 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 cifrado
// com a chave privada do LERA
OUT:      hashTlv data:
OUT:      lsrAddress = C0A80001, labelSpace = 0, hashDigest =
c_•E±_ó_ÈÒ¢ÞJoöŠ'™—¾³e¿„(™_;è„ëUÇ2_ð¸¥e%éš□yÏ”wÉ“>E_”_"£å6æèPáEÿ_&½ÌyíÁTÔ
„SÖÉî¥hÖ@ùyÃ_ƒÂÈäX,&Pm¼äü¬ñï…Œé*×å,>†é~SåÀYA_=%¶□.@©p
OUT: LABEL MAPPING MSG ***END***:
// tamanho da msg Label Mapping decodificada + cabeçalho LDP
OUT: Mesg 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 saida para a FEC
// 192.168.0.1
_ldp_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±_ó_ÈÒ¢ÞJoöŠ'™—¾³e¿„(™_;è„ëUÇ2_ð¸□¥e%éš□yÏ”wÉ“>E_”_"£å6æèPáEÿ_&½ÌyíÁTÔ
„SÖÉîYhÖ@ùyÃ_ƒÃËäX,&Pm□¼äü¬ñï…Œé*×å,>†é~SåÀYA_=%¶□.@©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 procesamento 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 26. 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 subseqüente):

```
--------------- 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
_ldp_global_add_outlabel
// instala na FIB o label recebido como label de saida 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 publica do LERA foi modificada propositalmente
// no arquivo ldp_peers do LERB (causando uma falha de autenticação), pois
// o LERB não pode decifrar 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 mensagen LDP Notification
OUT: LPD 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 27. 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.

6.7 Conclusões do Capítulo

Este capítulo apresentou os aspectos da implementação da autenticação fim a fim proposta ao LDP na plataforma linux. Foram descritas as principais alterações efetuadas no código fonte do pacote ldp-portable mostrando as partes de código inseridas. Foram apresentados também os passos necessários para a instalação, configuração e execução do ambiente de testes utilizado para verificar a implementação realizada e por final foram discutidos e demostrados os resultados obtidos na fase de implementação deste trabalho.

7 CONCLUSÕES

7.1 Revisão das Motivações e Objetivos

Espera-se com os resultados apresentados neste trabalho ter alcançado os objetivos levantados e justificado as motivações iniciais que serviram para alavancar esta dissertação.

7.2 Visão Geral do Trabalho

Este trabalho apresentou uma solução de autenticação para o protocolo LDP, de forma a viabilizar uma autenticação fim a fim, entre um LSR de ingresso e seu respectivo LSR de egresso, durante o estabelecimento de um novo LSP. Conforme verificado pelo levantamento dos trabalhos relacionados apresentados, não se tem conhecimento de uma solução semelhante que efetivamente atenda o propósito de autenticar, num escopo fim a fim, o estabelecimento de LSPs entre LSRs não-adjacentes no protocolo LDP. Dessa forma a solução deste trabalho é inédita dentro do seu escopo de aplicação.

No trabalho foram apresentadas as motivações, os objetivos gerais e específicos que situam o contexto desta dissertação e foi descrito como o trabalho foi organizado. Numa primeira etapa foram apresentadas as características, conceitos básicos e o funcionamento do MPLS e também as características básicas e o funcionamento do protocolo LDP. Foi mostrado como ocorre o processo de troca de mensagens, as fases de estabelecimento, manutenção e encerramento das sessões LDP e as operações de gerência com etiquetas MPLS (*labels*).

Numa segunda estapa foram apresentadas algumas ameaças as quais o LDP está vulnerável, discutiu-se maneiras de amenizar estas vulnerabilidades e foram apresentados alguns trabalhos relacionados a segurança do protocolo LDP. Após foi descrito com detalhes como ocorre o estabelecimento de um LSP entre LSRs não-

adjacentes, de modo a situar o contexto de aplicação da solução proposta no trabalho e a solução de autenticação fim a fim proposta para o LDP foi apresentada e discutida

Na última etapa do trabalho foram apresentados os aspectos da implementação da autenticação fim a fim, proposta ao LDP, realizada na plataforma Linux. Foram descritas as principais alterações efetuadas no código fonte do pacote ldp-portable, mostrando as partes de código inseridas, foram apresentados também os passos necessários para a instalação, configuração e execução do ambiente de testes utilizado para verificar a implementação realizada e por final foram discutidos e demostrados os resultados obtidos na fase de implementação deste trabalho.

7.3 Contribuições e Escopo do Trabalho

Conclui-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 a segurança do ambiente.

Uma forma de autenticação fim a fim, para viabilizar a autenticação mútua entre os LSRs de ingresso e egresso durante o estabelecimento de um LSP, é de fundamental importância para a segurança do protocolo LDP, especialmente dentro de ambientes MPLS multi-domínio, onde os domínios não são confiáveis entre si. Pôde-se constatar através da implementação realizada na plataforma Linux, que este objetivo foi prontamente atendido pela proposta de autenticação apresentada.

Como um exemplo clássico de ambientes multi-domínio, temos o provimento de VPNs baseadas em BGP/MPLS, em ambientes multiprovedores VPN, retratado em [ROSEN, 1999] e [ROSEN, 2002], onde vários provedores VPN fornecem o serviço VPN baseados em acordos que possuem entre si.

Embora não comprovado por um experimento prático, como uma implementação, a solução de autenticação apresentada neste trabalho pode ser perfeitamente aplicada ao protocolo CR-LDP, visto que o mesmo tem sua concepção baseada no LDP.

Em comparação aos trabalhos correlatos discutidos, a solução apresentada por [DE CLERCQ, 2001] possui em níveis gerais os mesmos objetivos da proposta deste trabalho. Através de uma análise aprofundada da proposta de [DE CLERCQ, 2001], conclui-se que a mesma apresenta um erro arquitetural, fato reconhecido pelos seus autores através de contato por e-mail. Por considerar erroneamente, que ao enviar uma mensagem solicitando o estabelecimento de um LSP para uma determinada FEC, o LSR de origem sabe qual é o LSR de destino que vai processar a requisição, o escopo de aplicação da solução apresentada por [DE CLERCQ, 2001], dentro do LDP, fica drasticamente reduzido e pode ser aplicada apenas a uma minoria de casos.

Em contrapartida, a solução de autenticação apresentada neste trabalho é completa e possui um escopo de aplicação genérico e abrangente. Dessa forma pode atender a todas as situações de estabelecimento de LSPs entre LSRs no LDP, embora sua principal aplicabilidade seja voltada a comunicações entre LSRs não-adjacentes.

7.4 Perspectivas Futuras

Para dar continuação a este trabalho, sugere-se como perspectivas futuras:

a) Especificar e implementar a distribuição de chaves usando certificação digital que foi abordada na seção 5.5.

b) avaliar os pós e contras de prover confidencialidade as informações transportadas pelo protocolo LDP. Existem opiniões que divergem neste sentido, dessa forma seria justificável um estudo aprofundado. Caso conclua-se vantajosa a adoção de confidencialidade no ambiente, sugere-se a especificação de uma solução que atenda as necessidades levantadas pelo estudo de viabilidades realizado.

c) analisar e propor as modificações necessárias na proposta de autenticação fim a fim apresentada neste trabalho, de modo a fornecer suporte a IP Versão 6 (IPv6). Não serão necessárias grandes mudanças para atingir este propósito, porém este objetivo não estava incluso no escopo do corrente trabalho, dessa forma ficou pendente para revisões futuras.

8 REFERÊNCIAS BIBLIOGRÁFICAS

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

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

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, 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.

BUDA, G.; CHOI, D.; et. al. **Security Standarts for the Global Information Grid**. IFIP/IEEE International Symposium on Integrated Network Management, Seatle, Maio de 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, 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, 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.

HEFFERNAN, A. **Protection of BGP Sessions via the TCP MD5 Signature Option**. Draft-ietf-idr-rfc2385bis-00.txt, março de 2002. Disponível por http://www.ietf.org/internet-drafts/draft-ietf-idr-rfc2385bis-01.txt. Acesso em 19 mar. 2002. (Trabalho em progresso).

JAMOUSSI, 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, 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, J; et. al. **Project: MPLS for Linux**. Início das atividades em 27 de novembro de 2000. Disponível por http://sourceforge.net/projects/mpls-linux. Acesso em 06 fev. 2002. (Trabalho em progresso).

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.

NEEDHAM, R.; ABADI, M. **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 ago. 2002.

Network Magazine, Ed. jan de 1999. **MPLS: A Progress Report**. Disponível por http://www.networkmagazine.com/article/NMG20000426S0015. Acesso em 15 mar 2002.

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, 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, 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.

ROSEN, E.; REKHTER, Y. **BGP/MPLS VPNs**. RFC 2547, Março de 1999. Disponível por http://www.ietf.org/rfc/rfc2547.txt. Acesso em 13 set. 2001.

ROSEN, E.; REKHTER, Y. **BGP/MPLS VPNs**. Draft-ietf-ppvpn-rfc2547bis-02, julho de 2002. Disponível por http://www.ietf.org/internet-drafts/draft-ietf-ppvpn-rfc2547bis-02.txt. Acesso em 08 ago. 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, 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, William. **Cryptography and Network Security: Principles and Pratice**. New Jersey, 1999, editora Prentice-Hall, Inc., 2ª ed.

WARD, D. **Securing BGPv4 using IPsec**. Draft-ward-bgp-ipsec-00.txt, fevereiro de 2002. Disponível por http://www.ietf.org/internet-drafts/draft-ward-bgp-ipsec-00.txt. Acesso em 12 mar. 2002. (Trabalho em progresso).

WU, Chun-Te; LIU,Huey-Ing; et. al. **Supporting Virtual Private Dial-Up Services Over Label Switching Based Networks**. IEEE/IFIP Network Operation and Management Symposium, Hawai, Maio de 2000.

ANEXO 1

Sequência de Passos Usada para a Instalação do Ambiente de Testes

Os programas fonte de todas as ferramentas foram armazenadas em /usr/src:

Passo 1) extrair o kernel do linux:

```
# cd /usr/src/
# tar -jvxf linux-2.4.19.tar.bz2
# ln -s linux-2.4.19 linux
# cd /usr/include
# mv linux linux.old
# mv asm asm.old
# ln -s /usr/src/linux/include/linux
# ln -s /usr/src/linux/include/asm
```

Passo 2) instalar o pacote mpls-linux:

```
# cd /usr/src/linux
# patch -p1 <../mpls-linux/patches/linux-kernel.diff
```

Passo 3) habilitar suporte ao MPLS no kernel e compilar:

```
# cd /usr/src/linux
# cat /usr/src/mpls-linux-1.1/QUICK.START
  Configure Linux kernel to turn on MPLS (atleast the following)
        make menuconfig (opcional ARCH=i386)
        Code maturity level options  --->
        [*] Prompt for development and/or incomplete code/drivers
        Networking options  --->
        [*] Network packet filtering (replaces ipchains)
          [*] Multi Protocol Label Switching - MPLS
          IP: Netfilter Configuration  --->
        <*> IP tables support (required for filtering/masq/NAT)
            <*> Packet mangling
            <*> MPLS target support
  NOTE: You need to turn on "Multi Protocol Label Switching" before
        you'll see "MPLS target support".

Compilar o kernel do Linux:
# make dep
# make clean
# make bzlilo
# make modules
# make modules_install
# vi /etc/lilo.conf
  image=/vmlinuz
  label=mpls-1170
  read-only
  root=/dev/hda3
```

```
# lilo
```

Reeiniciar a máquina e selecionar a compilação do kernel "mpls-1170" na inicialização da máquina.

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

```
# cd /usr/src
# tar -xvf ldp-portable-0.200.tar.gz
Obs: os fontes do zebra via comando "CVS" foram armazenados
/usr/src/zebra.
```

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

```
# cd /usr/src/zebra
# patch -p1 < ../ldp-portable/zebra-ldp.diff
# cd mplsd
# vi ./create-links  (acertar o path "SRC=/usr/src/ldp-portable/lib")
# chmod 755 ./create-links; ./create-links;
# cd /usr/src/zebra
# ./configure
# make
# make install
```

O comando "make install" gera por default os binários "zebra" e "mplsd" em /usr/local/sbin.

ANEXO 2

Passos para Configurar e Executar o LDP no Ambiente de Testes

Neste anexo são mostrados os comandos necessários a serem executados em cada um dos LSRs do ambientes de testes (Figura 20). Os arquivos executáveis do zebra (zebra) e do ldp-portable (mplsd) ficam localizados no diretório /usr/local/sbin e os respectivos arquivos de configuração "mplsd.conf" e "zebra.conf" ficam localizados no diretório /usr/local/etc.

A seguir são apresentados os arquivos de configuração e comandos necesários para executar o LDP no LERA e no LERB.

a) Configurando o LERA

Arquivo de configuração do ZEBRA (/usr/local/etc/zebra.conf):

```
!
! Zebra configuration saved from vty
!
hostname LERA
password zebra
log file log.zebra
!
interface lo
 description Loopback
 ip address 127.0.0.1/8
 ip address 192.168.0.1/32 secondary
 no shutdown
!
interface eth0
 description Ethernet0
 ip address 200.237.245.86/24
 no shutdown
!
interface eth1
 description Ethernet1
 ip address 11.0.1.1/24
 no shutdown
!
router-id 192.168.0.1
ip route 192.168.3.1/32 200.237.245.150
ip route 12.0.0.0/8 200.237.245.150
!
line vty
!
```

Arquivo de configuração do LDP-PORTABLE (/usr/local/etc/mplsd.conf):

```
Zebra configuration saved from vty
!
hostname LERA
password mplsd
!
interface eth0
 mpls ldp
!
interface lo
!
interface eth1
!
router zebra
!
mpls ldp
 lsr-id 192.168.0.1
 trace address
 trace binding
 trace debug
 trace error
 trace event
 trace general
 trace init
 trace label
 trace normal
 trace notification
 trace packet-dump
 trace packet
 trace path
 trace periodic
 trace policy
 trace route
 trace state
 trace task
 trace timer
!
line vty
!
```

Para iniciar a execução do "zebra" e "mplsd" no LERA são necessários os seguintes passos:

a) iniciar a execução do ZEBRA

```
# zebra -l -f /usr/local/etc/zebra.conf -P 2601 -d
```

b) iniciar a execução do LDP-PORTABLE (mplsd). Este comando armazena todas as saídas de DEBUG no arquivo "log".

```
# mplsd -l -f /usr/local/etc/mplsd.conf -P 2603 > log 2>&1 &
```

c) é necessário executar alguns comandos manualmente na interface de comandos do "mplsd" para que o LDP passe a operar, pois estes comandos são ignorados

durante a leitura do arquivo /usr/local/etc/mplsd.conf na inicialização do mplsd (pendências de implementação do LDP-PORTABLE):

```
# telnet localhost 2603
login: mplsd
LERA> enable             // entra no modo privilegiado
LERA> configure terminal // entra no modo de configuração de comandos
LERA> mpls ldp           // configuração global do LDP
LERA> lsr-id 192.168.0.1 // atribui o LSR-ID deste LSR
LERA> exit
LERA> interface eth0     // entra na configuração da interface ethernet0
LERA> mpls ldp           // habilita o LDP na interface eth0
LERA> exit
LERA> exit
LERA> logout
```

b) Configurando o LERB

Arquivo de configuração do ZEBRA (/usr/local/etc/zebra.conf):

```
! Zebra configuration saved from vty
!
hostname LERB
password zebra
log file log.zebra
!
interface lo
 description Loopback
 ip address 127.0.0.1/8
 ip address 192.168.3.1/32 secondary
 no shutdown
!
interface eth0
 description Ethernet0
 ip address 200.237.245.150/24
 no shutdown
!
interface eth1
 description Ethernet1
 ip address 12.0.1.1/24
 no shutdown
!
router-id 192.168.3.1
ip route 192.168.0.1/32 200.237.245.86
ip route 11.0.0.0/8 200.237.245.86
!
line vty
!
```

Arquivo de configuração do LDP-PORTABLE (/usr/local/etc/mplsd.conf):

```
Zebra configuration saved from vty
!
```

```
hostname LERB
password mplsd
!
interface eth0
 mpls ldp
!
interface lo
!
interface eth1
!
router zebra
!
mpls ldp
 lsr-id 192.168.3.1
 trace address
 trace binding
 trace debug
 trace error
 trace event
 trace general
 trace init
 trace label
 trace normal
 trace notification
 trace packet-dump
 trace packet
 trace path
 trace periodic
 trace policy
 trace route
 trace state
 trace task
 trace timer
!
line vty
```

Para iniciar a execução do "zebra" e "mplsd" no LERB são necessários os seguintes passos:

a) iniciar a execução do ZEBRA

```
# zebra -l -f /usr/local/etc/zebra.conf -P 2601 -d
```

b) iniciar a execução do LDP-PORTABLE (mplsd). Este comando armazena todas as saídas de DEBUG no arquivo "log".

```
# mplsd -l -f /usr/local/etc/mplsd.conf -P 2603 > log 2>&1 &
```

c) é necessário executar alguns comandos manualmente na interface de comandos do "mplsd" para que o LDP passe a operar, pois estes comandos são ignorados durante a leitura do arquivo /usr/local/etc/mplsd.conf na inicialização do mplsd (pendências de implementação do LDP-PORTABLE):

```
# telnet localhost 2603

login: mplsd
LERA> enable             // entra no modo privilegiado
LERA> configure terminal // entra no modo de configuração de comandos
```

```
LERA> mpls ldp           // configuração global do LDP
LERA> lsr-id 192.168.3.1 // atribui o LSR-ID deste LSR
LERA> exit
LERA> interface eth0     // entra na configuração da interface ethernet0
LERA> mpls ldp           // habilita o LDP na interface eth0
LERA> exit
LERA> exit
LERA> logout
```

ANEXO 3

Descrição do Funcionamento da Autenticação TCP baseada em MD5 definida para o LDP na RFC 3036

O protocolo LDP utiliza a autenticação TCP baseada em MD5 da seguinte maneira, conforme especificado na RFC3036 [ANDERSSON, 2001], sessão 2.9.2:

- O uso da autenticação TCP baseada em MD5 para conexões LDP é uma opção configurável no LSR.
- Um LSR que usa autenticação TCP baseada em MD5 é configurado com uma senha secreta (compartilhada) para cada par LDP em potencial.
- O LSR aplica o algoritmo MD5 conforme especificado na RFC2385 [HEFFERNAN, 1998] para gerar o *hash* MD5 do segmento TCP a ser enviado a um par LDP. A senha compartilhada e o segmento TCP servem de entrada para a função *hash*.
- Quando o LSR recebe um segmento TCP cujo ao qual foi aplicada a autenticação TCP baseada em MD5, o mesmo valida este segmento calculando a função *hash* MD5 daquele segmento TCP usando a sua chave secreta local e compara o valor resultante com o *hash* que foi passado junto com o pacote recebido. Se a comparação falhar, o segmento TCP é descartado sem envio de notificação alguma ao remetente.
- O LSR ignora HELLOS LDP de qualquer LSR para o qual uma senha não tenha sido configurada localmente. Isso assegura que o LSR estabeleça conexões TCP/LDP apenas com os LSRs para os quais uma senha tenha sido configurada.