Uma Implementação do MPLS para Redes Linux

Tomás A. C. Badan,* Rodrigo C. M. Prado, Eduardo N. F. Zagari,
Eleri Cardozo e Maurício Magalhães
DCA - FEEC - UNICAMP - Caixa Postal 6101
Campinas, SP, Brasil, CEP 13083-970
Email: {tomas,prado,zagari,eleri,mauricio}@dca.fee.unicamp.br

Sumário

A arquitetura MPLS (*Multiprotocol Label Switching*) é hoje um importante esforço de padronização em curso no IETF. Talvez a característica mais atrativa do MPLS é permitir uma implementação de roteamento baseado em restrições (CR). CR é um pilar para muitas funções de engenharia de tráfego visando introduzir qualidade de serviços nas redes IP. MPLS já é disponibilizado pelos principais fabricantes em roteadores e comutadores de alto desempenho. Visando a difusão da tecnologia MPLS, algumas implementações de domínio público estão disponíveis. Estas implementações se destinam a microcomputadores com sistema operacional tipo UNIX conectados através de rede de baixo custo como *Ethernet*. Este artigo descreve uma implementação do MPLS em curso na Faculdade de Engenharia Elétrica e de Computação (FEEC) da UNICAMP. Esta implementação se destina a suportar pesquisas em temas como qualidade de serviço, redes ativas e gerência de redes. A implementação apresenta alguns pontos de destaque tais como interface CORBA para gerência e especificação formal em UML (*Unified Modeling Language*).

Palavras-chave: Internet: protocolos, serviços e aplicações; MPLS; Qualidade de Serviço; Gerência de Redes; Engenharia de Tráfego.

Abstract

The MPLS (Multiprotocol Label Switching) architecture is today an important standardization effort conducted by the IETF. Perhaps the most attractive feature of MPLS is its ability to support a constraint-based routing (CR) implementation. CR is the foundation for many traffic engineering functions aimed to introduce quality of service in large IP networks. MPLS is already available in modern high-end routers and switches from the major network vendors. In order to widespread the MPLS technology, some public-domain implementations of MPLS are available. These implementations are targeted to microcomputers running UNIX-like operating systems and connected through low cost networks such as Ethernet. This paper describes an MPLS implementation being developed at the School of Electrical and Computer Engineering (FEEC) of UNICAMP. This implementation aims to support research in topics including

*Professor licenciado da Faculdade de Engenharia Elétrica da Universidade Federal de Goiás

quality of service, active networks, and network management. The implementation has some highlights such as a CORBA management interface and an UML (Unified Modeling Language) formal specification.
Keywords: Internet: protocols, services and applications; MPLS; Quality of Service; Network Management; Traffic Engineering.

1 Introdução

A utilização de procedimentos de engenharia de tráfego se constitui numa atividade essencial à operação de grandes redes IP. Engenharia de tráfego visa otimizar o uso de recursos de rede, minimizando o desbalanceamento de carga causado pelo roteamento IP convencional. Neste contexto, a padronização da arquitetura MPLS (*Multiprotocol Label Switching*) [4] em curso no IETF (*Internet Engineering Task Force*) é vista como de fundamental importância. MPLS permite o estabelecimento de caminhos comutados (LSPs - *Label Switching Paths*) através da rede IP nos quais a decisão de encaminhamento é baseada em um rótulo inserido em cada pacote. Estes caminhos, análogos a circuitos virtuais como em redes ATM e Frame Relay, permitem transportar determinada categoria de tráfego com qualidade de serviço.

Os principais fabricantes de roteadores IP disponibilizam MPLS em seus equipamentos de maior porte (*high-end systems*). Portanto, construir uma rede MPLS para experimentação é proibitivo dado seu alto custo. Visando difundir a tecnologia MPLS, algumas implementações destinadas a microcomputadores começaram a surgir. Dentre estas, cita-se os projetos conduzidos no NIST (NIST Switch) [1], atualmente disponível apenas em plataformas FreeBSD, e na Universidade de Wisconsin (MPLS for Linux) [2]. Estes projetos implementam MPLS para microcomputadores com sistema operacional tipo UNIX e conectados através de redes Ethernet. A implementação do MPLS para microcomputadores configurados como roteadores consiste de uma atualização (*patch*) do núcleo do sistema operacional e um protocolo para distribuição de rótulos (*labels*). A atualização do sistema operacional visa alterar o processamento do protocolo IP viabilizando a comutação por rótulo. O protocolo de distribuição de rótulos permite o estabelecimento de LSPs. RSVP-TE (*Resource Reservation Protocol - Traffic Engineering*) [6] e LDP (*Label Distribution Protocol*) [8] são os dois protocolos de distribuição de rótulos em padronização pelo IETF.

Uma avaliação destas implementações de domínio público do MPLS mostrou que:

- estão em constante evolução visando a correção de erros, a incorporação de novas funções e a aderência aos padrões;
- são difíceis de manter devido à deficiência de documentação e, em alguns casos, apresentam baixa robustez;
- são de difícil integração com outros sistemas de software como gerenciadores de políticas (*policy managers*), negociadores de banda (*bandwidth brokers*) e sistemas de gerência de redes;
- apresentam interfaces de configuração e gerência muito limitadas (via linha de comando).

Estas limitações motivaram-nos a desenvolver uma implementação do MPLS. Os principais requisitos desta implementação são:

- destinada exclusivamente ao sistema operacional Linux e redes *Ethernet*;
- projeto segundo práticas estabelecidas na engenharia de software;
- implementação *multithreaded* e orientada a objeto em C++;
- distribuição de rótulos através do protocolo LDP (escolhido com base na argumentação discutida em [9]);
- interface de configuração e gerência compatível com CORBA (Common Object Request Broker Architecture);
- alterações mínimas do núcleo do sistema operacional Linux.

A Figura 1 ilustra a arquitetura da nova implementação. Um *daemon* implementando o protocolo LDP executa no espaço do usuário (i.e., fora do núcleo do sistema operacional). O plano de encaminhamento consiste de uma atualização do núcleo do sistema operacional Linux visando adicionar capacidade de comutação por rótulo tanto nos roteadores de borda quanto nos roteadores de núcleo.

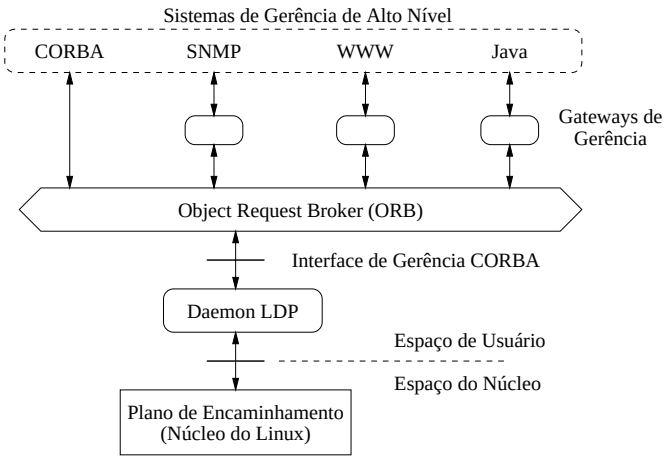

Figura 1: Arquitetura para uma nova implementação MPLS.

Esta nova implementação se destina ao ensino e à pesquisa. No ensino, a implementação permite incorporar MPLS no ensino prático de redes de computadores via utilização de recursos já existentes. Na pesquisa, a disponibilização de uma interface CORBA permite a interoperabilidade da implementação com vários sistemas de software que já incorporam CORBA, por exemplo, sistemas de gerência de redes e plataformas de agentes móveis. A implementação dará suporte às pesquisas em curso na FEEC/Unicamp nas áreas de engenharia de tráfego, redes ativas e gerência de qualidade de serviço.

Um fator que consideramos importante nesta implementação é seu processo de desenvolvimento. Adotamos um processo de desenvolvimento de software evolutivo e orientado a objeto[1], em que UML (*Unified Modeling Language*) [5] é utilizada nas fases de análise e projeto do software. Esta escolha se justifica pelas seguintes razões:

[1]Em oposição, por exemplo, a um desenvolvimento baseado em especificação formal.

1. Tanto a especificação MPLS quanto o núcleo do sistema operacional Linux estão em evolução, o que demanda uma constante atualização do software. Esta atualização é natural nos processos evolutivos e penosa nos processos baseados em métodos formais.
2. UML possui poder de expressão suficiente para modelar a dinâmica de um protocolo de rede complexo como LDP. UML favorece ainda a documentação do software.
3. Ferramentas CASE sofisticadas, como a utilizada neste desenvolvimento (Rational ROSE[2]) favorece o desenvolvimento com rapidez, baixo custo e confiabilidade.
4. Um desenvolvimento evolutivo permite, a partir de uma implementação básica, a adição incremental de funcionalidades.

Este artigo descreve a implementação do MPLS em curso na FEEC/Unicamp. A implementação básica já foi concluída e se encontra em fase inicial de testes. O artigo está organizado da seguinte forma. Na seção 2 é apresentada a especificação e a análise do sistema. Na seção 3, é proposta uma arquitetura para o mesmo. Nas seções 4 e 5 são detalhados, respectivamente, o Plano de Controle e o Plano de Encaminhamento. Na seção 6 é apresentado um exemplo de aplicação do sistema e, por fim, são discutidas aplicações futuras.

2 Especificação do Sistema

Os modelos representados pelos diagramas UML são utilizados para visualizar e documentar o desenvolvimento de sistemas. Nesta etapa inicial os documentos relacionados ao LDP [8] e ao MPLS [7] foram estudados buscando levantar os conceitos e requisitos definidos nas especificações.

O protocolo LDP corresponde a um conjunto de procedimentos e mensagens que permitem aos roteadores aderentes a esta especificação, ou LSRs (*Label Switching Routers*), estabelecerem LSPs através da rede. LDP associa a cada LSP criado uma classe (FEC - *Forwarding Equivalent Class*) para definição dos pacotes que serão mapeados naquele determinado LSP. Cada FEC é especificada como um prefixo de endereço de qualquer comprimento ou um endereço IP completo.

LSRs que utilizam LDP para trocar informação sobre rótulos e FECs são denominados Parceiros LDP (*LDP Peers*) e, para este fim, estabelecem entre si uma Sessão LDP (*LDP Session*).

A comunicação entre Parceiros 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, mais o Espaço de Rótulo (*Label Space*), que é definido dentro do escopo do LSR.

A especificação LDP define quatro categorias de mensagens que são trocadas entre os LSRs:

- mensagens de descoberta: utilizadas para anunciar e manter a presença de um LSR na rede;
- mensagens de sessão: usadas para estabelecer, manter e terminar uma sessão entre parceiros LDP;

[2]http://www.rational.com

- mensagens de divulgação: utilizadas para criar, alterar e remover mapeamentos de rótulos para FECs;
- mensagens de notificação: usadas para trocar informação e sinalizar erro.

Para estabelecer um LSP cada LSR deve: descobrir parceiros LDP na rede; estabelecer sessão com cada parceiro; enviar mensagem de requisição de rótulo para uma determinada FEC; e responder a solicitação com o mapeamento rótulo/FEC correspondente. Quando todos os LSRs do LSP obtiverem o mapeamento para a FEC, o LSP estará estabelecido.

A especificação CR-LDP[3] estende o protocolo LDP definindo um mecanismo de configuração fim-a-fim que permite o roteamento baseado em restrição, principalmente o roteamento explícito que corresponde a uma lista de LSRs compondo um LSP independentemente do roteamento convencional.

A arquitetura MPLS permite que um LSR distribua rótulo para FEC em resposta a uma requisição explícita de outro LSR. Este método é denominado *Downstream On Demand*. Existe ainda o método *Downstream Unsolicited* em que o LSR pode distribuir rótulos sem que tenha ocorrido uma requisição.

O controle de distribuição de rótulos pode ser independente - um LSR pode divulgar rótulos para seus vizinhos a qualquer momento que desejar - ou ordenado - um LSR somente pode iniciar a transmissão de mapeamento de rótulos para FECs para a qual ele já possui mapeamento ou quando for o roteador de borda (LER - *Label Edge Router*) de egresso.

O modo de retenção de rótulo define se o LSR deve ou não manter um mapeamento de rótulo para uma FEC obtido de um LSR que não é o próximo *hop* para a FEC. Ele pode ser conservador ou liberal. No modo conservador são retidos apenas o mapeamento do próximo *hop* de acordo com o roteamento. No modo liberal são mantidos todos os mapeamentos obtidos dos parceiros LDP.

A especificação LDP define ainda uma opção configurável para a detecção de laço. Este mecanismo previne que uma mensagem de requisição de rótulo percorra os LSRs da rede fechando um laço.

Dentre as opções de gerência e distribuição de rótulos, restingimo-nos, em função dos nossos objetivos, a um subconjunto da especificação. Escolhemos o método *Downstream On Demand*, o modo de controle de distribuição ordenado, o modo de retenção conservador e não fazemos detecção de laço.

A Figura 2 apresenta o diagrama UML de caso de uso. O administrador e operador da rede atuam no sistema, respectivamente, para configurá-lo e criar/remover LSPs. O sistema interage com os sistemas localizados nos outros roteadores para estabelecer os LSPs e, também, encaminhar pacotes através do roteamento convencional.

A Figura 3 apresenta o diagrama de classe UML em uma perspectiva conceitual, obtido após a análise da especificação LDP.

A classe `AdmInterface` serve como ponto de acesso para a aplicação responsável pela configuração e gerência da rede MPLS. É esta classe que expõe uma interface CORBA.

A classe `LSR` é uma abstração do roteador como um todo e possui somente uma única instância. Sua função principal é coordenar as ações locais para o estabelecimento de LSPs. Esta atividade está centrada nesta classe pois ela se relaciona tanto com a interface anterior (*up-*

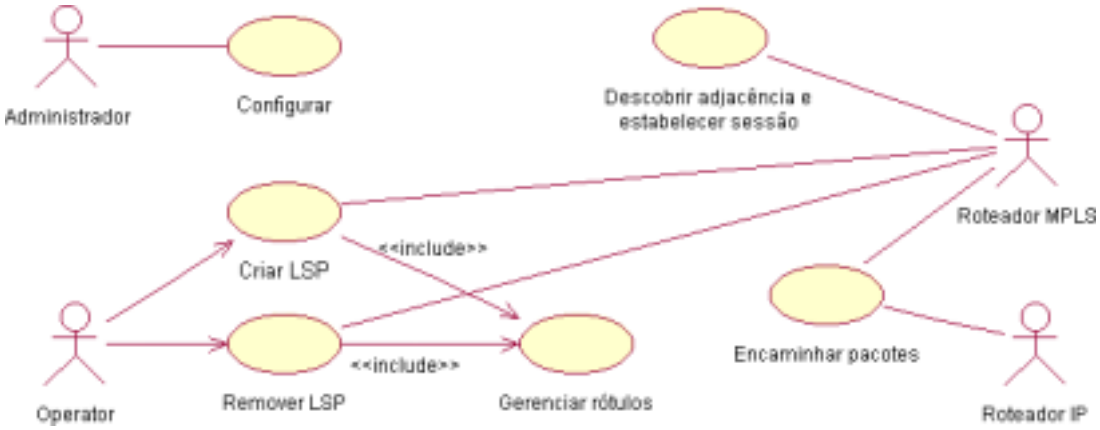

Figura 2: Diagrama de caso de uso MPLS.

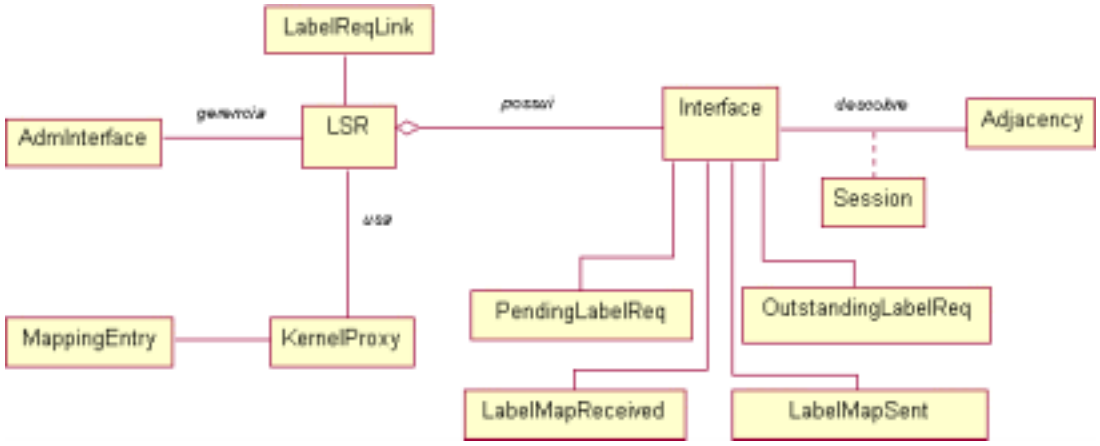

Figura 3: Diagrama de classe conceitual.

stream) quanto com a interface posterior (*downstream*)[3]. A classe `LabelReqLink` é utilizada para armazenar informação sobre a associação entre uma requisição de label recebida ainda pendente e a requisição enviada para o LSR que é o próximo *hop*.

A classe `Interface` encapsula as atividades relacionadas à interface de rede. Ela é responsável por receber e enviar mensagens de requisição e mapeamento de rótulos. Além disso, deve manter registros das mensagens. Para este fim utiliza as classes `PendingLabelReq`, `OutstandingLabelReq`, `LabelMapReceived` e `LabelMapSent`.

A classe `Adjacency` representa os outros roteadores parceiros LDP com relação a um determinado LSR.

A classe `Session` é responsável pelo estabelecimento e pela manutenção da sessão entre LSRs parceiros. Ela classifica e trata as mensagens trocadas entre os LSRs.

A classe `KernelProxy` abstrai as funcionalidades fornecidas pelo núcleo e também possui uma única instância. O mapeamento entre rótulo de entrada e rótulo de saída de um LSP, realizado por `KernelProxy`, é armazenado por intermédio da classe `MappingEntry`.

[3]Os termos anterior e posterior dizem respeito ao fluxo de pacotes em um LSP, os pacotes viajam de um LSR anterior para um LSR posterior

3 Projeto do Sistema

Na fase de projeto evidenciou-se a necessidade da separação do sistema em dois planos funcionais. O primeiro, Plano de Controle, consiste de um *daemon* executando no espaço de usuário e tem por função a criação e remoção dos LSPs entre os LSRs. O segundo, Plano de Encaminhamento, executa no espaço do núcleo e é responsável pelo suporte à comutação por rótulos.

O plano de controle é composto de três subsistemas:

1. Mecanismo de Descoberta, responsável por descobrir adjacências e manter relações ativas com as mesmas (potenciais parceiros LDP);
2. Estabelecimento e Manutenção de Sessão, responsável por estabelecer e manter sessões LDP com as adjacências descobertas para dar suporte à comunicação entre parceiros LDP;
3. Distribuição e Gerência de Rótulos, responsável pela alocação, requisição e divulgação de rótulos envolvidos na criação e remoção de LSPs.

A gerência de cada subsistema é feita através de um ponto de acesso, onde é disponibilizada a interface através da qual o administrador da rede MPLS pode ajustar parâmetros de configuração do sistema e/ou o operador pode solicitar a criação/remoção de LSPs.

A comunicação entre *daemons* de LSRs adjacentes e/ou parceiros LDP para a troca de mensagens de descoberta, de sessão e de divulgação é feita através de um ponto de acesso à camada de transporte (Ponto de Acesso para “Adjacência/Parceiro LDP”).

O plano de encaminhamento é dividido em dois subsistemas. O primeiro adiciona novas funcionalidades ao núcleo enquanto o segundo atualiza os componentes já existentes para suportar MPLS. No atual estado da implementação, a alteração mais substancial ocorre no componente IPv4.

A comunicação entre o plano de encaminhamento e o plano de controle ocorre via uma interface bem definida através das chamadas de sistema `ioctl` e `socket`.

A agregação destes componentes pode ser vista na Figura 4.

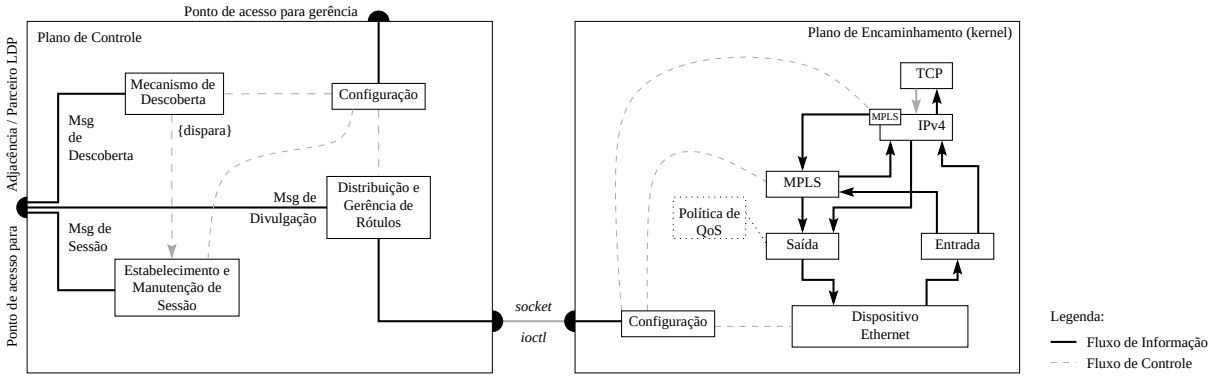

Figura 4: Interação entre os componentes do sistema.

4 Plano de Controle

O plano de controle desempenha basicamente três atividades: descoberta de adjacências, estabelecimento e manutenção de sessões e distribuição e gerência de rótulo. O sistema, quando da sua inicialização, deve ainda encontrar e identificar as interfaces de rede do equipamento que estiverem configuradas para trocar rótulos. Cada interface de rede passa a ser representada por uma instância da classe `Interface`.

4.1 Modelamento Dinâmico

Mecanismo de Descoberta

O mecanismo de descoberta permite a um LSR descobrir potenciais parceiros LDP. Existem duas variantes deste mecanismo: o básico, usado para descobrir LSRs vizinhos que estejam diretamente conectados no nível de enlace, e o estendido, para descobrir LSRs que não estejam diretamente conectados no nível de enlace.

No mecanismo básico, o LSR envia periodicamente mensagens LDP *Hello* como pacotes UDP para o endereço *multicast* do grupo "todos os roteadores desta sub-rede" em porta pré-definida (646). Já no mecanismo estendido utiliza-se o endereço de uma máquina para envio de mensagens *Hello*. O recebimento de uma mensagem LDP *Hello* identifica uma "Adjacência *Hello*" como um potencial parceiro LDP. A mensagem *Hello* identifica também o espaço de rótulo que esta adjacência deseja usar e o intervalo de tempo de validade da mensagem. Se este intervalo expirar, a adjacência é encerrada.

A relação entre a interface do LSR que recebe a mensagem de *Hello* e a interface da adjacência é caracterizada através do conceito de "papel". Comparando os endereços de rede IPv4 das duas interfaces como dois inteiros de 32 bits, define-se o papel de cada uma delas no estabelecimento da sessão LDP: ativo para o LSR com o maior endereço e passivo para o de menor endereço.

Estabelecimento e Manutenção de Sessão

O mecanismo de descoberta, isto é, a troca de mensagens LDP *Hello* entre dois LSRs dispara o estabelecimento de uma sessão entre eles. Este processo é realizado em duas etapas: estabelecimento de uma conexão de transporte e inicialização da sessão.

O comportamento do LSR no estabelecimento da conexão de transporte depende do papel que este desempenha. O LSR ativo toma a iniciativa de estabelecer a conexão TCP na porta LDP padrão (646) enquanto o LSR passivo fica aguardando conexões nesta porta.

A inicialização da sessão envolve a troca de mensagens de inicialização entre os dois LSRs. Esta mensagem especifica valores pretendidos para os parâmetros da sessão. Os parâmetros são, por exemplo, tamanho máximo de PDU, disciplina de divulgação de rótulo e método de detecção de laço.

O não recebimento de uma mensagem em um período de tempo determinado, *KeepAlive*, indica finalização da sessão. Portanto, para manter a sessão cada LSR deve enviar periodicamente PDUs LDP ao seu parceiro. Se não houver nenhuma informação a ser comunicada o

LSR envia uma mensagem de *KeepAlive* apenas para manutenção da sessão.

As atividades relacionadas à sessão estão encapsuladas na classe `Session`. Um objeto desta classe é instanciado ao final do estabelecimeto da conexão de transporte e passa por diferentes estados no seu ciclo de vida. A Figura 5 ilustra o diagrama de estados de um objeto da classe `Session` durante o processo de estabelecimento de uma sessão com um parceiro LDP [8].

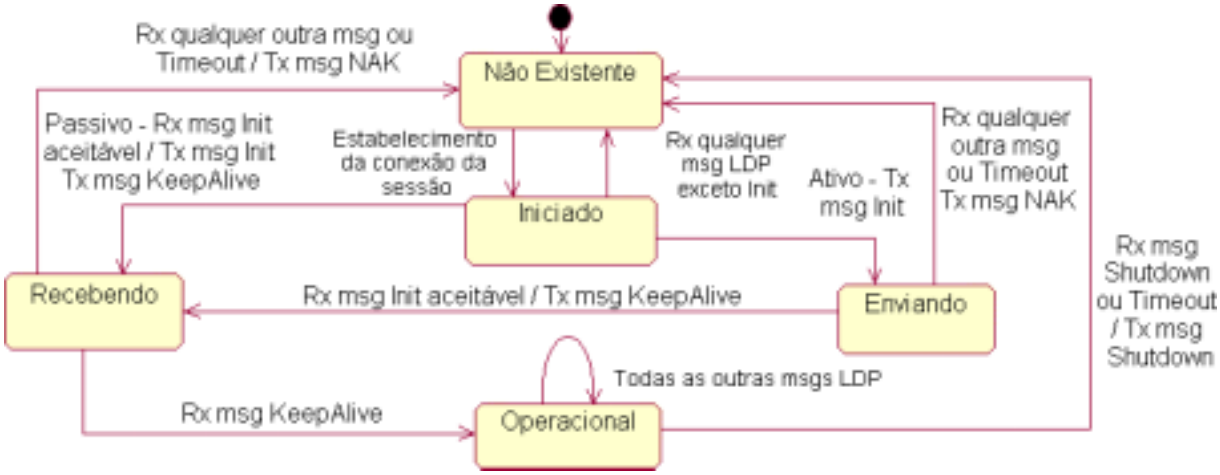

Figura 5: Diagrama de transição de estados de um objeto da classe `Session`.

Distribuição e Gerência de Rótulos

Após ter sido criada a sessão que irá tratar da troca de informações entre dois parceiros LDP, é necessário dar suporte à distribuição e gerência de rótulos. Como adotamos a técnica de distribuição de rótulos *Downstream On Demand*, com controle ordenado e modo de retenção conservador, esta atividade consiste em realizar a montagem de um novo LSP e a remoção de um LSP já existente.

A montagem de um novo LSP se dá a partir de uma solicitação do operador da rede MPLS no roteador de borda (LER) de ingresso, que dispara a seqüência de mensagens de solicitações de rótulos aos LSRs posteriores (*downstream*) no caminho até o LER de egresso para a FEC em questão.

Quando a solicitação de rótulo alcança o LER de egresso, um novo rótulo é alocado e instalado junto ao núcleo do roteador e uma mensagem de mapeamento de rótulo é enviada ao roteador anterior (*upstream*). Este por sua vez instala o rótulo recebido, aloca um novo rótulo e envia uma nova mensagem de mapeamento de rótulos ao LSR anterior. O processo se repete até atingir o LER de ingresso.

O processo de remoção de LSPs é similar ao de montagem diferindo basicamente no tipo de mensagem enviada ao parceiro LDP. Ao receber uma mensagem de liberação de rótulos o LSR procede a remoção do rótulo junto ao núcleo e envia uma nova mensagem de liberação de rótulo ao próximo parceiro LDP do caminho até o LER de egresso para a FEC.

A Figura 6 mostra o diagrama UML de seqüência para o recebimento de uma mensagem de

requisição de rótulo. Este diagrama é útil pois podemos observar a linha de vida dos objetos das classes envolvidas nesta atividade e a interação entre estes objetos.

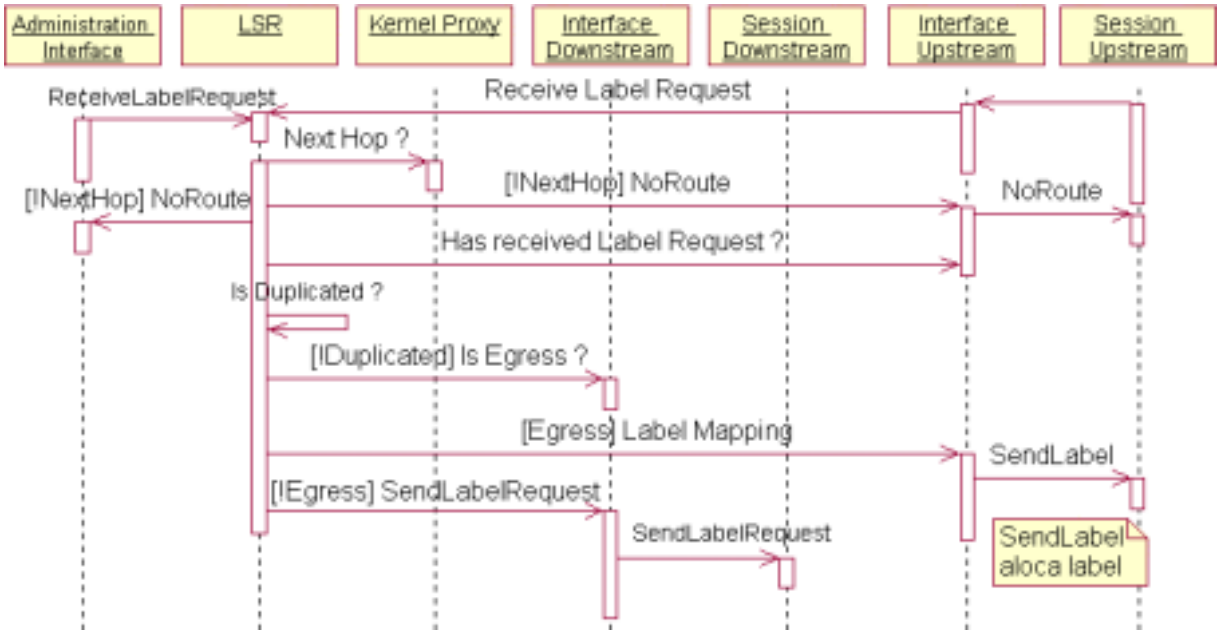

Figura 6: Diagrama de seqüência para mensagem de requisição de rótulo.

4.2 Implementação

Para atender aos requisitos de paralelismo das atividades do plano de controle, decidiu-se por uma implementação *multithreaded*. A escolha da linguagem de programação `C++` foi devido à facilidade de integração com o plano de encaminhamento, escrito em Linguagem `C`, no núcleo do Linux.

Após realizar a localização de interfaces, a *thread* principal do sistema deve ainda prover mecanismos para o recebimento e o envio de mensagens LDP *Hello* de e para suas adjacências e para o estabelecimento de sessões com estas. Isto é feito através da criação de novas *threads* específicas para cada função.

Para a realização do mecanismo de descoberta, existem duas *threads*. A primeira é responsável pelo envio periódico de mensagens LDP *Hello* através de cada interface configurada para troca de rótulos. A segunda deve aguardar mensagens LDP *Hello* e identificar sua interface de origem, caracterizando seu "papel" na relação de adjacência. Ambas *threads* estão associadas ao objeto `LSR` descrito no diagrama de classes da Figura 3.

Caso a interface que recebeu a mensagem tenha o papel ativo, cria-se uma nova *thread*, que irá cuidar do estabelecimento da sessão do lado do LSR ativo. Esta nova *thread*, ao contrário das demais, é temporária, ou seja, é criada para o estabelecimento de uma sessão e terminada quando a sessão se torna operacional. É responsabilidade desta *thread* promover as transições de estados do objeto `Session`, cujo comportamento durante a fase de inicialização da sessão é mostrado o diagrama de estados da Figura 5.

O mecanismo de estabelecimento de sessão conta, ainda, com uma *thread* criada pela *thread*

principal para o estabelecimento de uma conexão de transporte, no caso do papel do LSR na sessão ser passivo. Nesta linha de execução, também associada ao objeto `LSR`, aguarda-se a solicitação de conexão vinda da adjacência com papel ativo e, estabelecida a conexão, cria-se uma nova *thread*, também temporária, para a incialização da sessão. Desta forma, a cada nova sessão estabelecida por um LSR, é criada apenas uma das duas *threads* temporárias, dependendo de seu papel na relação de adjacência ser ativo ou passivo. Em qualquer um dos casos, após estabelecida a sessão, é criada uma *thread* do mesmo tipo para cada novo objeto `Session`. Este objeto é o responsável pelas atividades de manutenção das sessões e pelo processamento de suas mensagens de distribuição e de gerenciamento de rótulos (mensagens de divulgação).

5 Plano de Encaminhamento

O núcleo do Linux é monolítico devido ao seu projeto original [10, 11]. Isto significa que qualquer funcionalidade, ou característica do sistema, que necessite ser executada no contexto do núcleo deve ser agregada ao arquivo de execução do núcleo no momento da compilação do código fonte[4]. Assim, uma das primeiras diretivas do projeto do MPLS foi modificar o código fonte (em nosso caso, usamos a versão 2.4.2) para provê-lo com as habilidades necessárias para lidar com o roteamento/encaminhamento baseado em rótulos.

Existem dois tipos de modificações que devem ser conduzidas no código fonte do núcleo do Linux. A primeira consiste em criar novo código que permita lidar com os novos componentes que estão sendo inseridos, enquanto a segunda consiste em alterar o código existente a fim de provê-lo com habilidades de reconhecer os novos componentes e manuseá-los adequadamente. Estes dois procedimentos são detalhados nas subseções que se seguem.

5.1 Adições ao Núcleo

Nesta etapa, o objetivo é proporcionar ao núcleo a habilidade de lidar com pacotes marcados como MPLS e permitir que programas que executam no contexto do usuário possam configurar e obter informações das estruturas de dados do MPLS adequadamente.

Normalmente, uma interrupção é gerada quando o dispositivo de rede informa ao sistema que há um quadro necessitando de tratamento. Após a troca de contexto, da interrupção para o núcleo, a tarefa para este tratamento se inicia. Vamos chamá-la aqui de “discriminador” por conveniência. Uma de suas atribuições é enviar o quadro para cada componente interessado em recebê-lo. A escolha do componente é baseada no tipo do protocolo de camada 3 que o quadro transporta e componentes registram este interesse no momento da inicialização do sistema.

Neste momento, o discriminador passa o controle para o componente escolhido, o qual inicia o processamento do pacote. Duas decisões podem ser tomadas: ou o pacote é deixado no sistema, ou o pacote deverá ser encaminhado para uma ou mais interfaces de saída. Caso a última decisão seja tomada, o pacote é adicionado à fila de saída da interface de rede e ações relevantes à qualidade de serviço, se existir alguma, são aplicadas sobre esta fila, antes que o pacote seja enviado para a rede. Este fluxo de informações pode ser visto na Figura 7.

[4]Módulos, por outro lado, são adicionados ao núcleo em tempo de execução.

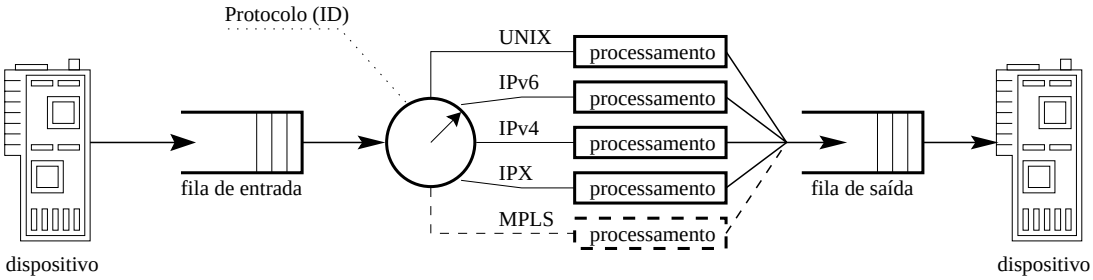

Figura 7: Alteração realizada no fluxo de processamento de pacotes no núcleo do Linux para suportar pacotes do tipo MPLS.

Portanto, a primeira providência é informar ao núcleo que os pacotes do tipo MPLS não devem ser descartados, mas desviados para uma função que irá tratá-los adequadamente. Assim, o primeiro passo é registrar esta necessidade para o sistema no momento do *boot*, dizendo ao discriminador qual função deverá ser chamada. Este fluxo de informações está representado na Figura 7 através de linhas tracejadas.

O algoritmo usado para processar os pacotes que entram no sistema marcados como MPLS é ilustrado na Figura 8.

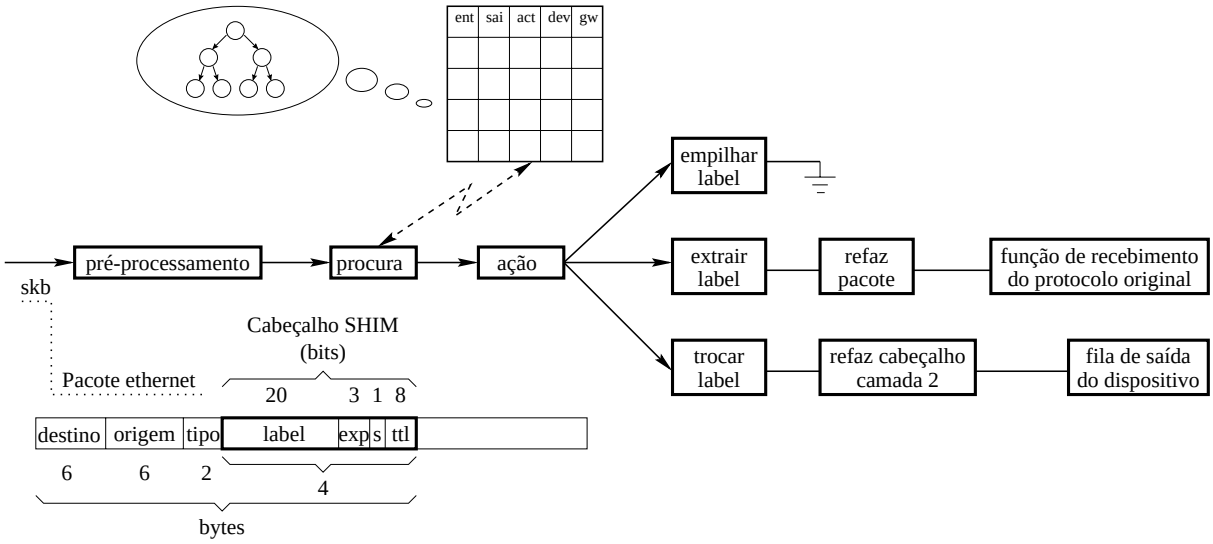

Figura 8: Processamento de pacotes pelo MPLS.

Deve-se observar o formato dos pacotes na entrada deste componente, o qual será o mesmo para os pacotes que saem do sistema. Como estamos lidando somente com a tecnologia *Ethernet*, o encapsulamento adotado é o cabeçalho SHIM que possui um tamanho de quatro bytes e sua posição no pacote é após o cabeçalho *Ethernet* e antes do cabeçalho correspondente aos protocolos de camada 3. Destes quatro bytes, 20 bits são alocados para o rótulo, 3 bits são reservados, 1 bit é para informar se o cabeçalho é o último de uma pilha de rótulos e 8 bits possuem a mesma interpretação do campo **TTL** do cabeçalho IP.

A primeira operação a ser realizada é um pré-processamento sobre o pacote, cujo objetivo

é a detecção de situações de exceção, como por exemplo, se o dispositivo, no qual o pacote chegou, possui um espaço de rótulos associado. Esta é uma informação importante, pois nos informa o endereço de início da tabela que armazena os rótulos do sistema. Qualquer situação de erro encontrada nesta etapa ou nas outras que se seguem geralmente resulta em descarte do pacote.

Com o rótulo do pacote sendo usado como indexador, realiza-se uma pesquisa em uma tabela de rótulos para obter as informações que regem as ações a serem tomadas. A tabela de rótulos é implementada através de uma árvore binária auto-balanceável. A chave de procura é o rótulo de chegada e as informações armazenadas se resumem a: qual operação a ser realizada no pacote, qual o rótulo de saída, qual o dispositivo de saída e, para o protocolo IPv4, qual o próximo *hop*.

Como esta tabela é usada também na etapa de configuração do sistema, o seu acesso é regulamentado através de um *lock* do tipo leitura e escrita.

Após determinada o tipo de ação, uma de três etapas podem ser aplicadas:

- Extrair rótulo: o rótulo é retirado do pacote. Se for o último, ou seja, após o cabeçalho SHIM vier o cabeçalho da camada 3, o pacote é refeito sem o cabeçalho SHIM e entregue para a função de entrada de camada 3 adequada. Neste caso, a máquina está atuando como um roteador de egresso. Por outro lado, se for um rótulo pertencente a uma pilha de rótulos o pacote é descartado pois esta funcionalidade não está implementada;
- Empilhar rótulo: o pacote é descartado (funcionalidade não implementada);
- Trocar rótulo: um novo rótulo é inserido no pacote e este enviado para o *buffer* da placa de saída. Neste caso, a máquina está atuando como um roteador de núcleo da rede.

Finalmente, o último componente é o que permite a configuração de todo o sistema. Existem três tipos de ações:

- Configurar um dispositivo: incluir e excluir um espaço de rótulos;
- Configurar a tabela de rótulos: adicionar e retirar rótulos e ligar (*bind*) um rótulo de entrada a um de saída;
- Configurar as FECs de entrada: adicionar e retirar FECs e ligar uma FEC de entrada a uma de saída. Uma FEC somente pode ser adicionada se estiver presente na tabela de roteamento. Somente FECs pertencentes ao protocolo IPv4 foram implementadas.

Esta tarefa (contexto do núcleo) está sob o controle do protocolo LDP (contexto do usuário) e a troca de contexto ocorre através de uma chamada de sistema (*system call*) `ioctl`.

A função `ioctl` opera baseada em um descritor de arquivo, o qual é retornado através de uma chamada de sistema – `socket`. Assim, para manusear as chamadas à função `ioctl`, foi criada uma nova família de protocolo, chamada de AF_MPLS a qual é passada como parâmetro para o função `socket`.

O usuário pode obter informações do sistema, também, através do sistema de arquivos *proc*. Através dele é possível verificar a tabela de rótulos, as FECs presentes no sistemas e quais os dispositivos possuem espaço de rótulos e o seu valor.

5.2 Alteração do Núcleo

Dentre algumas alterações realizadas no código do núcleo do Linux, a principal recai sobre o componente IPv4, o qual permite que a máquina opere como roteador de ingresso.

O componente IPv4 possui duas tabelas de roteamento, uma gerada dinamicamente e outra estática, chamada de tabela de encaminhamento (*FIB table*). Isso é realizado para aumentar a eficiência na procura de rotas.

A tabela dinâmica contém entradas com as rotas já descobertas. Esta tabela é indexada pelo endereço de destino do pacote e só é permitido casamentos exatos durante uma pesquisa.

Caso não encontre uma entrada na tabela dinâmica, o sistema realiza uma procura mais lenta, na tabela de encaminhamento. Se o sistema conseguir estabelecer uma nova rota, esta é adicionada na tabela dinâmica e é utilizada posteriormente para o envio de pacotes. Finalmente, as entradas da tabela dinâmica são temporizadas e retiradas após um determinado intervalo de tempo. Esta operação pode ser vista na Figura 9.

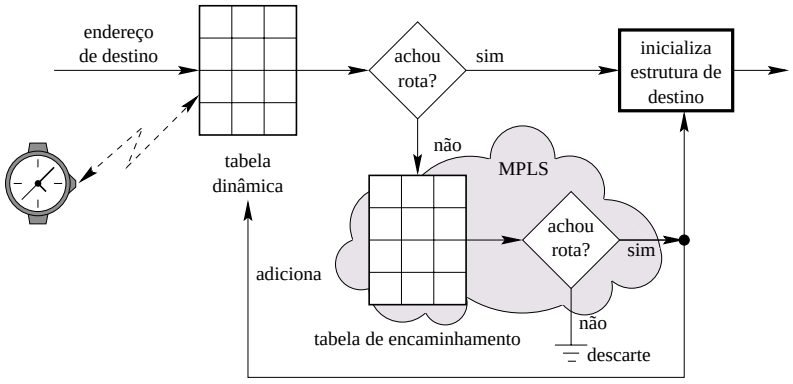

Figura 9: Fluxo de informações no roteamento IPv4.

A alteração realizada neste componente foi para permitir a identificação de uma FEC pelo MPLS. Isto é obtido inserindo mais um campo que descreve uma entrada na tabela de encaminhamento, ou seja, informações relativas ao próximo *hop* para o MPLS. Assim, sempre que uma entrada na tabela de encaminhamento tiver uma informação de rótulo associada, os pacotes serão desviados para o componente MPLS.

Outras alterações, de menor porte, foram realizadas para permitir que informações pudessem fluir de um componente a outro dentro do sistema.

6 Exemplo de Aplicação

Para testar o plano de encaminhamento, foi realizado um experimento simples envolvendo sete máquinas, conforme ilustrado na Figura 10. A idéia é verificar se as máquinas estão desempenhando corretamente o seu papel na rede, ou seja, se estão atuando como roteadores de ingresso, núcleo e egresso, além verificar como o MPLS pode ser útil na engenharia de tráfego.

Segundo esta topologia, qualquer máquina situada na rede 10.1.1.0/24 irá se comunicar com as máquinas situadas na rede 10.1.4.0/24 através dos roteadores IP A, B e D e, para o caminho

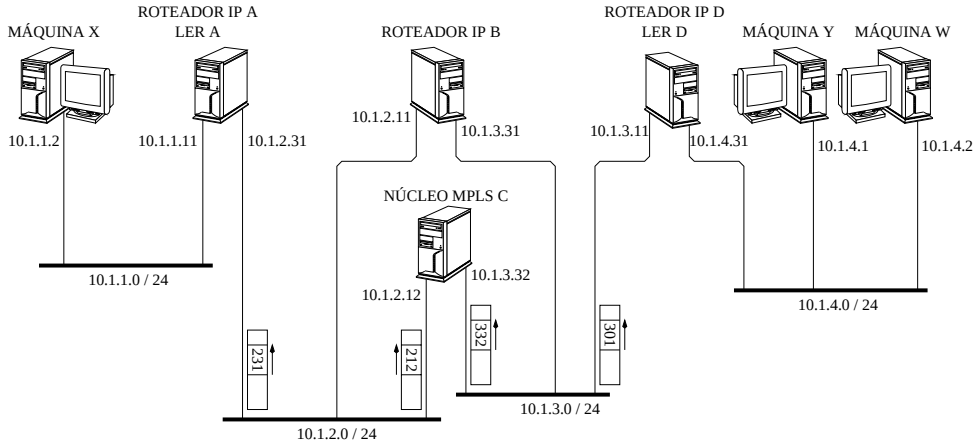

Figura 10: Topologia de rede utilizada no exemplo.

reverso, pelos roteadores IP D, B e A. Neste exemplo, criamos um LSP entre a máquina X e a máquina W e vice-versa, usando os roteadores MPLS A, C e D.

Considerando o LSP formado no sentido de fluxo dos pacotes da máquina X para a máquina W, o LER A está atuando como ingresso, o LSR C como núcleo e o LER D como egresso.

Para comprovar a existência do LSP, utilizou-se o comando `ping`. A metodologia empregada consiste em realizar um `ping` entre as máquinas X e W antes e depois de estabelecer o LSP. Assim, verificamos que nos dois casos as máquinas se comunicam conforme o previsto. Contudo, ao desabilitar o MPLS de um dos LERs que compõe o LSP, voltando este a atuar como um roteador IP, verificamos que o comando `ping` pára de funcionar. Conclui-se, portanto, que o roteamento realmente é realizado utilizando MPLS.

Utilizamos, também, o programa `ethereal`[5] em cada interface de rede de cada roteador para visualizar o conteúdo dos pacotes e verificar a presença do rótulo.

Um segundo resultado que podemos extrair deste experimento é a possibilidade de se realizar engenharia de tráfego. Com o uso do MPLS, podemos estabelecer caminhos alternativos para o fluxo de pacotes apenas criando novos LSPs, como os dois criados entre as máquinas X e W. Desta maneira, no caso de um fluxo de pacotes entre as máquinas X e W, a rota estipulada pelo protocolo IP será ignorada pelos LSRs. Assim, teremos um caminho diferente daquele seguido pelo fluxo de pacotes entre as máquinas X e Y, distribuindo a carga entre os roteadores B e C.

O comando `traceroute` foi utilizado para verificar este comportamento. Isto porque as máquinas roteadoras IP usadas no experimento foram habilitadas para gerar e responder às requisições ICMP, ao contrário daquelas usadas como roteadores MPLS, as quais não implementam esta funcionalidade. Assim, os pacotes gerados por este comando que seguem o caminho IP terão todos os roteadores listados em sua saída, enquanto que os pacotes que seguirem pelo MPLS não terão os roteadores de núcleo listados na saída. Desta forma, ao utilizarmos o comando `traceroute` associado ao analisador de pacotes `ethereal`, podemos constatar a distribuição de carga entre os roteadores B e C, na comunicação entre as máquinas X - Y e X - W, respectivamente.

[5]htpp://www.ethereal.com/

7 Conclusões

A implementação do MPLS descrita neste artigo possui algumas características que julgamos importantes. Primeiramente, acreditamos que um desenvolvimento evolutivo e orientado a objeto é plenamente factível (apesar de raro) para a implementação de protocolos de rede. Outro ponto importante se refere à gerência da rede. Apesar da gerência SNMP ser bem estabelecida, outras interfaces de gerência devem ser igualmente disponibilizadas. Nesta direção, disponibilizamos uma interface CORBA para a gerência da implementação MPLS aqui descrita. Esta interface nos permitirá pesquisar novas estratégias de gerência, por exemplo, estratégias baseadas em agentes móveis. Finalmente, o domínio do código fonte nos permitirá evoluir, portar, especializar e aprimorar a implementação.

Conforme mencionado no artigo, a implementação MPLS descrita dará suporte ao ensino e à pesquisa. Em termos de pesquisa, três áreas são de especial interesse: qualidade de serviço (QoS), redes ativas e gerência avançada de redes. Estamos desenvolvendo um *framework* de suporte à QoS para redes MPLS onde funções de engenharia de tráfego viabilizarão o estabelecimento de fluxos com qualidade de serviço. Uma aplicação importante nesta linha são as redes privadas virtuais (VPN) com QoS assegurada. Na linha de redes ativas e gerência avançada de redes, a interface CORBA disponibilizada em nossa implementação permitirá acoplar plataformas de agentes móveis (por exemplo, Voyager). Os agentes controlam e gerenciam a operação da rede, por exemplo, criando LSPs por demanda, reconfigurando parâmetros do MPLS e viabilizando uma gerência pró-ativa da rede.

Referências

[1] http://www.antd.nist.gov/nistswitch/.

[2] http://nero.doit.wisc.edu/mpls-linux/.

[3] B. Jamoussi et al. Constraint-Based LSP Setup using LDP. Internet Draft, July 2000.

[4] U. Black. *Multi-Protocol Label Switching*. Prentice Hall, December 2000.

[5] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language User Guide*. Addison-Wesley Pub Co, 1998.

[6] D. Awduche et al. RSVP-TE: Extensions to RSVP for LSP Tunnels. Internet Draft, August 2000.

[7] E. C. Rosen et al. Multiprotocol Label Switching Architecture. Internet Draft, July 2000.

[8] L. Andersson et al. LDP Specification. Internet Draft, August 2000.

[9] L. Li et al. IP Traffic Engineering Using MPLS Explicit Routing in Carrier Networks: Between the Signaling Approaches - CR-LDP and RSVP. Whitepaper, April 1999. http://www.nortelnetworks.com/products/announcements/mpls/collateral/whitepaper.html.

[10] M. Beck et al. *Linux Kernel Internals*. Addison-Wesley Longman, second edition, 1998.

[11] A. Rubini. *LINUX Device Drivers*. O'Reilly, 1998.