LiME : A Linux based MPLS Emulator

Abhijit Gadgil and Abhay Karandikar

Department of Electrical Engineering,
Indian Institute of Technology, Bombay.
Powai, Mumbai - 400076.
gabhijit,karandi@ee.iitb.ac.in

ABSTRACT

In this paper, we present the design of a Linux based Emulator for Multiprotocol Label Switching (MPLS) network–LiME. LiME has been designed to emulate real world network toplogies and test actual MPLS protocol implementations unlike network simulators that would only provide an abstraction of the protocol. Moreover, LiME can used as a Protocol Development Environment by those developing MPLS based protocols. The design of LiME leverages on a multithreaded version of Label Distribution Protocol (LDP) and a switching engine in Linux kernel developed by us.

1. INTRODUCTION

Network layer routing consists of two components– forwarding and control. The forwarding component makes use of the information available in Forwarding Information Base (FIB) to forward packets along the data path. The control component usually consists of one or more distributed protocols and is responsible for populating the FIB. In traditional routing, the control component may be one or more routing protocols like OSPF, BGP.

Multiprotocol Label Switching (MPLS) [1] is a new network layer routing paradigm where the forwarding component is a Label Swapping algorithm. A label is a short fixed length shim header inserted in the packet by a router at the ingress node (called Label Edge Router (LER)) of a MPLS network. Each router in the MPLS domain (called a Label Switch Router (LSR)) uses the label as an index in the forwarding table, determines the forwarding treatment to be applied (including the next hop to be forwarded) and swaps the incoming label with another label (See [2, 3] for a full appreciation of label swapping and switching mechanism).

At an LER, the packets are classified into a set of Forwarding Equivalence Classes (FEC) that determine the treatment these packets will receive in the MPLS domain. The control component of MPLS creates bindings between labels and FEC and informs other Label Switched Router (LSR) of these bindings. The control component utilizes the information from routing protocols also. Various control protocols for exchanging label binding information are being standardized. These include Label Distribution Protocol (LDP) [4] and extensions to Resource Reservation Protocol (RSVP-TE) [5].

The task of testing these distributed protocols is very involved as it necessitates a network test bed. Network simulators like ns2 [6] provide a good framework for testing these protocols but are not useful for testing actual implementation as these simulators provide only abstraction of the protocol. Another approach would be to emulate the network in software. The emulator is different from a simulator in the sense that it provides Applications Programming Interface (API) for plugging in actual implementation thereby enabling a network device (called Device Under Test (DUT)) or a protocol to be tested for its performance in real network scenarios. Previous researchers have considered network emulator for TCP/IP based networks [7]. To the best of our knowledge, however, no emulator is available for MPLS networks.

The objective of this paper is to design and develop a Linux based Emulator for MPLS–LiME. One of the primary contributions of the paper is a scalable architecture of the emulator. The emulator can emulate user defined topologies. It would be able to support and tear down Label Switched Paths (LSP) based on topology changes. It also emulates the forwarding behavior of LSR and LER. For emulating the control protocol, the emulator leverages on a multi-threaded version of Label Distribution Protocol [8] developed by us. Another unique contribution of the design is the capability of LiME to inject MPLS traffic into the external world.

Rest of the paper is organized as follows. Section 2 discusses the top level design of LiME. The implementation aspects are covered in the following two sections. Section 5 focuses on the interaction of LiME with Linux TCP/IP stack. We conclude the paper in Section 6.

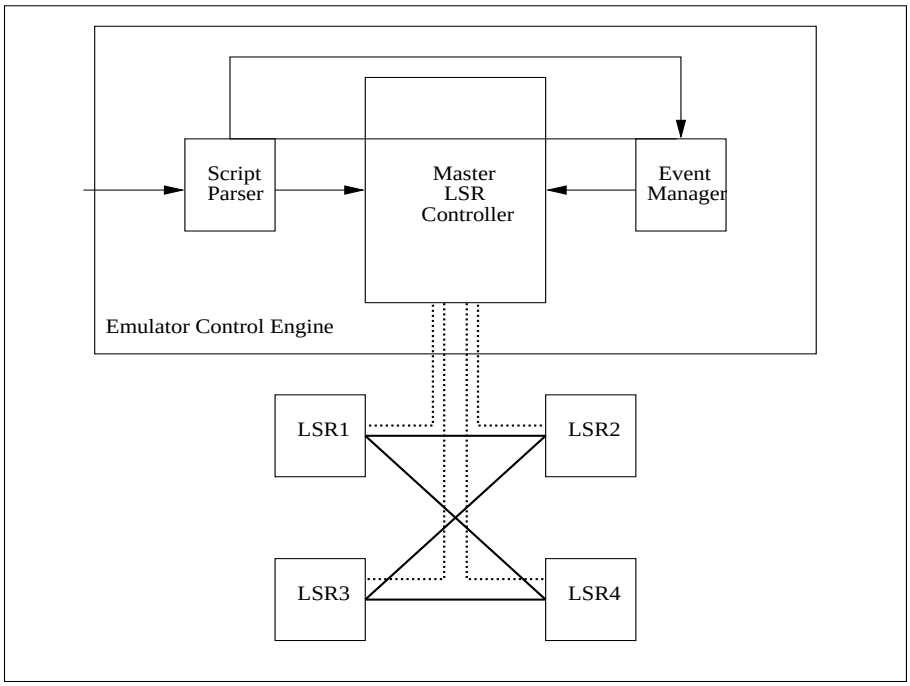

Figure 1: *MPLS Emulator Overview.*

2. TOP LEVEL DESIGN

One of the challenges in the design of LiME is to develop a scalable architecture. This would mean that the memory and computational requirements should grow as linearly as possible with the size of the network to be emulated. With this in mind, the following important design decisions were observed while developing the top level architecture of LiME.

1. Though LiME has been designed to work with Linux, it makes little changes to the existing network stack of Linux.

2. In order to avoid overheads of copying data to user–space and back to kernel–space while forwarding, the packets are switched in Linux kernel itself in LiME.

3. In LiME, we avoid creating multiple copies of the networking stack per emulated LSR.

4. The user interface is in the form of Tcl/Tk scripts.

The architecture of LiME consists of two main functional blocks (See Figure 1) - Emulator Control Engine and Emulated LSR/LER. The Emulator Control Engine (ECE) provides the abstraction for the network subsystem. The network topology to be emulated is given as an input to the control engine. The ECE has three main functional sub blocks. These are Script Parser, Master LSR Controller and Event Manager. The Emulated LSR (ELSR) performs the forwarding and control behavior of an instance of the LSR to be emulated. An Emulated LSR implements FIB, LDP and one or more MPLS devices attached to the LSR. The detailed designs of the ELSR and ECE are discussed in the following sections.

In addition to the ECE and ELSR, MPLS forwarding engine and Label Distribution Protocol as in [8, 9] and OSPF implementation and OSPF simulator environment as described in [10] also form part of LiME.

3. DESIGN OF EMULATED LSR

In LiME, an Emulated LSR comprises of an instance of control component, Forwarding Information Base and one or more MPLS devices. We briefly discuss each of them.

3.1. LSR/LER Control Component - (LDP/Routing)

An instance of LDP [4] will form the control component of emulated LSR. LDP is developed as a multi-threaded application in user-space of Linux employing TCP/UDP sockets. The LDP design has the following logical threads

1. The I/O thread - This thread implements the

socket based I/O operations of the LDP.

2. The Processing thread - This thread undertakes the processing functions of the received LDP packets, and maintaining the overall state of the LDP.

3. Kernel Interface thread - This thread forms the interface between LDP and the switching engine. It is based upon netlink sockets that facilitates the exchange of information between user-space and kernel-space.

These threads are co-ordinated and scheduled by a scheduler in the main loop. The FEC information required by the LDP is made available through OSPF.

3.2. Forwarding Information Base (FIB)

The packet label is used as an index into FIB to retrieve the corresponding entry for making a forwarding decision. The FIB implementation comprises of

- Next-Hop Label Forwarding Entry (NHLFE)
  `struct nhlfe_type` - It has information which is useful in actual forwarding of packets and outgoing label stack etc.

- Incoming Label Map(ILM)
  `struct ilm_entry_type` - A label space associated with a device, determines the way in which a labeled packet is interpreted. Two packets that have the same label values but belong to different label spaces will be treated differently. The ILM entry specifies incoming label entry in the label space associated with the device.

- FEC To NHLFE table (FTN)
  `struct fec_type` - FTN is used to make forwarding decision on unlabeled packets.

3.3. MPLS Device.

MPLS device is implemented as a loadable kernel module in the Linux kernel. The device provides the functionality of receiving packets and forwarding them on the data path. Upon initialization, MPLS device registers itself with the list of kernel network devices. One advantage of introducing the feature of MPLS Device is that it obviates the need for sending data path packets to user-space and processing them in the user-space thereby avoiding a data copy from kernel-space to user-space and back. Moreover, the approach enables to exploit the MPLS forwarding engine available in the kernel [9] directly without duplicating this functionality in the user-space.

3.4. Packet Forwarding in Emulated LSR

The following pseudo-code explains how a packet received at an MPLS device in the ELSR is processed.

```
Is the packet Labeled?
      /* ethertype == ETH_P_MPLSUC ? */
if (yes) {
      /* mpls_rcv */
    look for ILM entry in the label-space
    for the device.
      /* search_ilm_entry */
    if (entry exists){
      /* mpls_send */
        forward packet using NHLFE.
    } else {
        if default NHLFE exists,
        forward packet using default
        NHLFE or drop the packet.
    }
} else {
      /* ip_rcv */
   Send the packet to IP module.
   look for FTN entry for the packet in
   the FTN table for the LSR.
      /* fec_prefix_match */
   if(entry exists) { i
      /* grab_nhlfe_for_fec */
       forward using NHLFE.
      /* mpls_send */
   } else {
       Forward the packet using
       IP forwarding.
      /* ip_forward */
   }
}
```

Figure 2 depicts an emulation of two LSRs and their interface with the external IP/MPLS network.

4. EMULATOR CONTROL ENGINE (ECE)

An instance of LSR/LER provides the functions of a single MPLS node. The network abstraction of an MPLS network, comprising the emulated LSRs in a specified topology, is achieved through the Emulator Control Engine. The engine provides abstraction in the form of network topology and network events. The actions of multiple instances of LSRs, their configurations and performance monitoring are implemented in ECE. It has three main functional blocks- Tcl Script Interpreter, Event Manager, Master LSR controller.

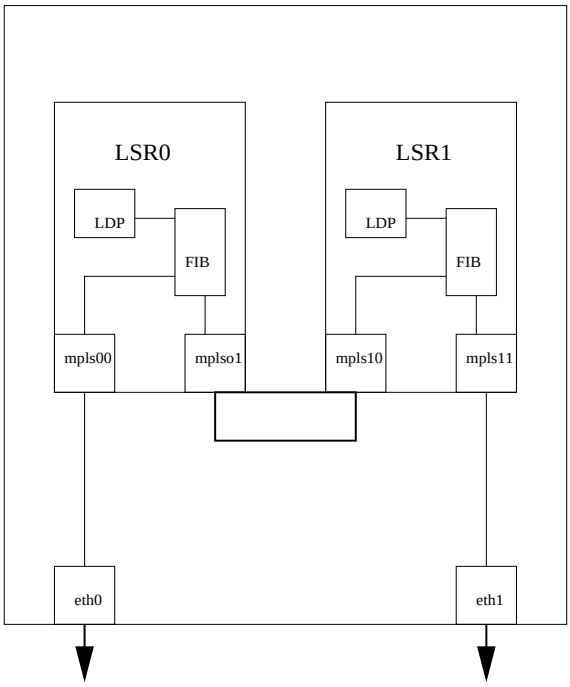

Figure 2: *Emulated LSR - Simple Topology.*

4.1. Tcl Script Interpreter.

Tcl script is used for specifying the network topology. The Tcl interpreter parses the input scripts and generates the topology information to be used by the master LSR controller. Various network events to be emulated like link failure, node failure are dispatched to the event manager.

4.2. Event Manager

The response of a DUT can be observed for different types of network events like - link between a pair of LSRs going down, disabling MPLS functionality on one of the LSRs etc. These network events are registered with the event manager via Tcl script. The event manager synchronizes these events with the global timer (described below) and dispatches them for execution to the Master LSR controller.

4.3. Master LSR Controller

Master LSR Controller is the heart of the emulator control engine. The important functions like timer management, resource management, LSR/LER scheduling are handled by this component. The following components achieve this functionality.

- Emulator Control Socket– Upon initialization, the master controller creates a server socket.This socket is used for exchange of information with the emulated LSR/LERs. An LSR/LER instance creates its own client socket, which talks to the server socket. However, this communication is different from that of MPLS control exchange facilitated through TCP/UDP sockets created by the LDP instance.
- Emulator Master Timer– One global timer is initialized while starting the emulator engine. All events in the emulator are synchronized with respect to this global timer. We define a "tick" in the timer to be 1/10th of a second, which is a reasonable resolution for exchanging control informations. However it should be noted that, this "tick" will not slow down the data path forwarding because, data path forwarding takes place in the kernel-space.
- The scheduler - After the initialization and instantiation of LSRs is over, the scheduler is invoked which schedules the events sequentially. The scheduler performs the function of resource management in the emulator. The management and integrity of respective data structures is handled by particular instance of LSR/LER. Following tasks are scheduled in the decreasing order of their priority.
  - ELSR control protocol tasks.
  - Event management tasks
  - ELSR kernel interface tasks
  - Tasks related to maintaining state of ECE.
  - ELSR configuration and notification tasks.

4.4. Emulator Main Loop

The function of LiME can be explained by the following pseudo code. The LiME main loop can be described as follows.

```
Input topology information.
    /* get_nw_topology */
Open the server socket.
    /* create_ctrl_sock */
Create the timer thread.
    /* init_timer */
Create an instance of LSR/LER for every
LSR in the given topology.
    /* for (each lsr) { init_lsr } */
Send the configuration information on
the control socket.
   /* send_config_info */
while(true) {
```

```
    Schedule LSR events. /* schedule */
}
```

5. LINUX TCP/IP STACK AND APIS

We now discuss the issues related to Linux kernel while implementing the MPLS device and a set of APIs for the protocol developers which will be exposed through LiME implementation.

LiME private `ioctls` –

The standard mechanism to configure network devices uses the `ioctl` system call. We extend the basic functionality in the kernel to provide some configuration options for control protocol, label space and encapsulation mechanisms used with a MPLS device.

Handling Unlabeled Packets in LiME–

At any given MPLS device, an unlabeled packet will be received if the device is attached to an LER, or if the packet is a control protocol packet or if the packet is delivered to this LSR after Penultimate Hop Popping (PHP). Thus the decision for unlabeled packet may differ for every device even for the same destination address. We exploit the multiple routing table mechanism of Linux, to forward packets based on the input device on which the packet is received. A rule is set to look into a specific forwarding table depending upon the input device index. Each MPLS device will have a unique index in the Linux kernel in LiME.

LiME Interface to External Network–

MPLS device acts as a wrapper for actual system devices like `eth0, ppp0, atm0` (for Ethernet, PPP and ATM interfaces) etc. Packets received on these devices will be handled by the MPLS device receive function.

LiME Applications Programming Interface–

LiME APIs are library functions that can be used by a programmer to port any MPLS software on LiME platform. These API calls essentially provide necessary hooks to the ECE. This enables LiME to be used as a Protocol Development Environment.

6. DISCUSSION

In this paper, we have described a scalable architecture of LiME- a Linux based MPLS Emulator. LiME can be used as a test-environment for testing control protocol implementations. The APIs also enable LiME to be used as a Protocol Development Environment. The framework is useful in studying the deployment effects of different traffic engineering algorithms within operation MPLS networks. Some of these applications are currently being investigated by the authors. The given architecture can be enhanced further to incorporate a distributed ECE. This would facilitate emulation of very large topologies. The emulator can also be extended to support other control protocols like RSVP-TE [5], Differentiated Services over MPLS [11] etc. These issues will be considered in future releases for LiME.

REFERENCES

[1] E Rosen, A Viswanathan, R Callon, "Multiprotocol Label Switching Architecture.", January 2001, http://www.ietf.org/rfc/rfc3031.txt

[2] B Bruce and Y Rekhter, "MPLS: Technology and Applications", Morgan Kaufman, 2000.

[3] E Grey, "MPLS: Implementing the Technology", Addison Wesley, 2001.

[4] L Anderson, P Doolan, N Feldman, A Fredette, B Thomas, "LDP Specification", January 2001, http://www.ietf.org/rfc/rfc3036.txt

[5] D Awduche, L Berger, Der-Hwa Gan, T Li, V Srinivasan, G Swallow, "RSVP-TE : Extensions to RSVP for LSP Tunnels", work in progress., August 2001. http://www.ietf.org/internet-drafts/draft-ietf-mpls-rsvp-lsp-tunnel-09.txt

[6] The Network Simulator ns-2, http://www.isi.edu/nsnam/ns/

[7] X Huang, R Sharma and S Keshav, "The Entrapid Protocol Development Environment", *Proceedings of INFOCOM, 1999.*

[8] A Gadgil, "Label Distribution Protocol (LDP) Design Document-v1.0" Technical Report, EE Dept, IIT Bombay, February 2001. (Available on request from authors.)

[9] R Soni, "Architecture of MPLS Forwarding Engine in Linux Kernel", MTech Thesis, IIT Bombay, January 2001.

[10] J Moy, "OSPF Complete Implementation", Addison Wesley 2001.

[11] F Le Faucheur, L Wu, B Davie, S Davari, P Vaananen, R Krishnan, P Cheval, J Heinanen, "MPLS Support for Differentiated Services", work in progress., April 2001, http://www.ietf.org/internet-drafts/draft-ietf-mpls-diff-ext-09.txt