USER-MODE LINUX BASED MPLS EMULATOR

Balachander.R *and P.* ***Venkataram***

Dept. of Electrical Communications Engineering,
Indian Institute of Science, Bangalore - INDIA.
E-mail: **rbalachander@ece.iisc.ernet.in, pallapa@ece.iisc.ernet.in**

ABSTRACT

Multi-Protocol Label Switching ***(MPLS) is*** *a key technology able to enhance IP networks with advanced Traffic Engineering and* ***QoS*** *functionalities. It* ***intro-****duces a connection-oriented forwarding plane based on a label switching paradigm.*

Development of network applications for commercial purposes over the Internet requires extensive verification and validation in large networks. Real networks are however commonly limited in ***size,*** *number of network devices,* ***and*** *in topology configurations supported. Such limitations make large-scale verification and validation tests hard to accomplish.*

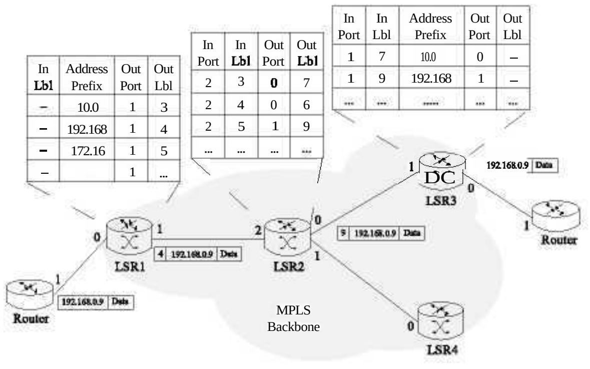
By emulating a ***network,*** *the size, router specifications, and topology can be controlled. This allows for more flexible and larger scale testing than in a real* ***network.*** *Also, costs are reduced since no expensive network devices are needed.*

This paper presents design and implementation procedures for User-Mode Linux ***(UML)*** *based MPLS network emulator. The designed emulator is useful for testing on real-time the MPLS network planned to implement in an organization with a specified network traffic. The designed emulator has been tested with standard test suite under various traffic conditions.*

1. INTRODUCTION

MPLS is a technology that integrates the label-swapping paradigm with network-layer routing **[1]. Each** MPLS packet has a label. Depending on different Layer 2 and Layer 3 technologies **involved,** different label encoding schemes **can** be used.

Label swapping is done by associating labels with routes and using the label value in the packet forwarding process. Packets are classified and routed at **the** ingress Label Switching Routers **(LSR's)** of an MPLS-capable domain. The mapping between IP packets and a Label Switched Path **(LSP)** is done by providing a Forwarding Equivalence Class **(FEC)** specification for each LSP. MPLS labels are then **inserted.** When an LSR receives a labeled **packet,** it will use die label as an index to look up the forwarding table. This is faster than the process of parsing the routing table and search for the longest match done in IP routing. The packet is processed as specified by the forwarding table entry. The incoming label is replaced by the outgoing **label, and** the packet is switched to the next **LSR.** Before a packet leaves an MPLS **domain,** its MPLS label is re**moved.** The MPLS operation procedure in a sample network is shown in Figure **1.**



In Lbl	Address Prefix	Out Port	Out Lbl
–	10.0	1	3
–	192.168	1	4
–	172.16	1	5
–		1	...

In Port	In Lbl	Out Port	Out Lbl
2	3	0	7
2	4	0	6
2	5	1	9
...	...	...	...

In Port	In Lbl	Address Prefix	Out Port	Out Lbl
1	7	10.0	0	–
1	9	192.168	1	–
...	...	......	...	...

Fig. **1.** MPLS Operating Procedure

In MPLS, mere are three signaling protocols for exchanging label binding information are being standardized. These are Label Distribution Protocol (LDP), Constraint-based Routed Label Distribution Protocol (CR-LDP) and extensions to Resource **ReSerVation Protocol (RSVP-TE).**

The **task** of testing these distributed protocols is very involved as it necessitates a network test bed. Network simulators like **ns2** 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 there by enabling a network device (called Device Under Test **(DUT))** or a protocol to be tested for its performance in real network **scenarios.The** strength of network emulators is that researchers can easily move network code between **a** real network and an emulated network. Furthermore, me code being tested can run on any type of system without modification and any testing and analysis software will work **in** both environments.

MPLS network emulators are expected to reproduce **the** unfavorable conditions of MPLS network and WANs in a controllable and **repeatable** lab setting.

User-Mode Linux (UML) is a patch for the Linux kernel, allowing an executable binary to be built and executed within user space on a Linux system.

The objective of this paper is to design and develop an Emulator based on User-Mode Linux for MPLS Networks. The proposed emulator reproduces the unfavorable conditions of MPLS network with RSVP-TE signaling and recreates the dynamic behavior of the Intranet such that network model can be reproduced including those models that change with Routing, diffserv with traffic conditioning.

The organization of the rest of the paper is as follows. Section 2 gives an introduction of UML. Section 3 discuses some of the existing works. Section 4 discusses proposed UML based MPLS emulator. The emulation results are covered in section 7. Finally, we conclude the paper in Section 8.

2. USER-MODE LINUX

User-mode Linux (UML)[2] is the port of the Linux kernel to Linux. It implements a Linux virtual kernel running on a Linux host Its hardware is virtual, being constructed from resources provided by the host UML can run essentially any application that can run on a host

The standard layer diagram of a Linux operating system is shown in Figure 2 . At the bottom is hardware with an operating system controlling it. The OS is broken into generic code, which is architecture-independent, and the architecture layer and drivers, which implement the hardware-specific functions needed by the generic kernel. Above that is the system call layer and user-level processes (like *process1).*

The user-mode kernel is implemented at this layer (see Figure 2, but it is structured exactly like the host kernel. It has a generic kernel layer, which may be exactly the same code as in the host kernel, with the architecture-specific functions implemented by a user-mode architecture layer and drivers, which are implemented in terms of system calls.

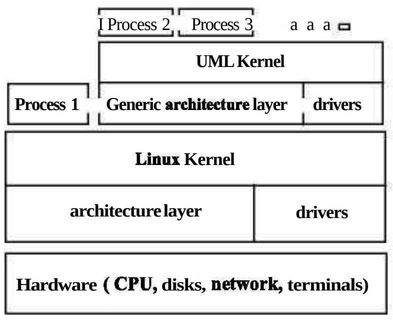


Fig. 2. User-Mode Linux Architecture

3. RELATED WORK ON MPLS EMULATOR

A LINUX based MPLS Emulator (LiME)[4] is an MPLS emulator that can be used as an test environment for testing control protocol implementations. LiME supports only LDP and it forward traffic between its emulated nodes using a kernel space MPLS stack. It is capable of running code developed for real network devices and may also communicate via a network interface with real MPLS devices.

Another MPLS emulator is the commercial NetTest MPLS InterEMULATOR[9]. InterEMULATOR is intended for testing of new protocols, network devices and to study algorithm behavior. It also allows active insertion of errors in packets in order to study error detection and correction. This emulator allows validation of performance, functionality and interoperability of network devices and algorithms.

4. PROPOSED UML BASED MPLS EMULATION

The proposed design has used some of the features of the RSVP-TE daemon from Teqila project[8]. The main components of the emulator are UML kernel's which acts as MPLS nodes, communications interfaces which will configure UML kernel's as virtual network and the rsvp daemon for running RSVP-TE signaling protocol.

4.1. Initializations

During the initialization MPLS features are enabled in each UML kernel, and a script is run for configuring the *dummy* interfaces to create a virtual network between the UML kernels. Then UML kernels will be booted on the host kernel and *rsvp* daemon will be run on all the UML kernels. The daemon supports Diffserv allowing to differentiate the forwarding behavior based on the value of EXP field of the MPLS header. It also supports for IntServ where resources are explicitly allocated on a per-LSP level. Traffic can be mapped on the LSPs very flexibly based on the destination address, protocol, destination ports and port ranges of IP packet.

4.2. MPLS Emulator Architecture

The overall architecture (see Figure 3) consists of a number of components both in user space and kernel space. The important parts of the kernel that are used, are MPLS support, *netfilter* to classify the packets, QoS and fair queuing to support differentiating between flows.

The prime user space component is the RSVP daemon that is responsible for RSVP signaling and maintenance of the MPLS state. The daemon is responsible for allocating and installation of MPLS labels during LSP set up and freeing and removing labels on LSP tear down.

Two components use the RAPI: *rtest* and *rapirecv* . *rtest* is an application that takes LSP requests and issues them to the daemon. *rapirecv* is an application that receives label requests at the egress and dictates the daemon to send a response back to the ingress. *rtest2* and *rapirecv_auto* (not shown in the figure) are extended version of *rtest* and *rapirecv* respectively that

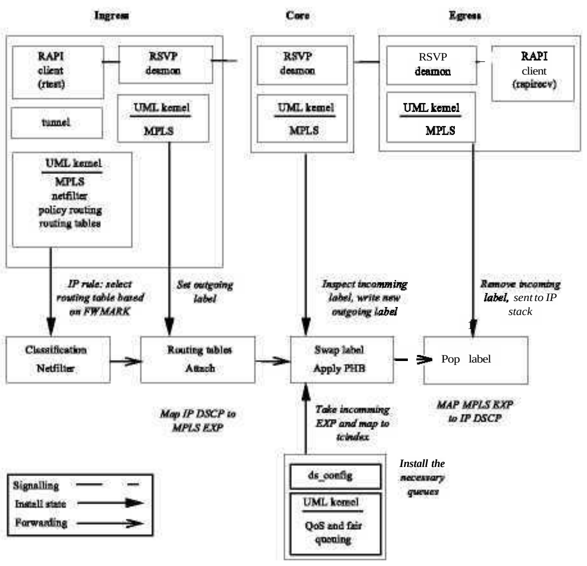


Fig. **3.** Overall Architecture of MPLS using **RSVP-TE** under UML

support the automatic set up of a (large) number of LSPs.

The *tunnel* is an application that maps traffic on an existing LSP. Mapping **can** be based on destination address, protocol, destination ports **and** port ranges (TCP **and** UDP). So basically anything that ***netfilter*** supports. Tunnel sends **status** packets requests **(mstat** packets) to the RSVP daemon in order to receive information about the installed state in the daemon (existing sessions, labels, reservations etc.). For example mstat packets are used to trace an LSP or to automatically send a **RESV** message at the egress.

43. The forwarding path in Emulator

In the ingress the packets are classified with *netfUter* (refer to Figure **3).Packets** are filtered on the OUTPUT and **PREROUTING** chain of the mangle table. The mangle table is needed because **the** *fwmark* needs to be set The OUTPUT and PREROUTING chains **are** used in order to filter on both locally generated and incoming traffic. Based on the value **of the** *fwmark* a routing table is selected (using policy routing). In the resulting routing table there is a MPLS tunnel interface acting as the default gateway. The tunnel interface encapsulates the packets on the LSP by attaching the correct outgoing label. The incoming DSCP is mapped to the EXP field in the MPLS header. A limitation of this architecture is that ***netfilter*** can only write (mark) on the mangle table and that only a single mark operation is possible. So while we can map traffic on the LSP also setting the DSCP at the same node is impossible. The solution is to set the DSCP before the traffic enters the ingress LSP.

In the core node the MPLS stack inspects the incoming label and sets the new outgoing label and next hop. At the **DiffServ** level the current EXP value of the packet is inspected. There is a mapping from this EXP value to a ***tcindex.*** The ***tcindex*** in **turn** determines the correct outgoing queue so the correct forwarding behavior **(PHB)** can be applied. Finally in the egress the incoming EXP field is mapped to the DSCP field and then the MPLS header is stripped off and the packet is sent to the IP layer.

5. CONFORMANCE TESTING OF MPLS

We have addressed the challenges of protocol **conformance** testing by developing a **conformance** test suite. While supporting some protocols protocols overall, it contains several test cases to validate routers for MPLS label encapsulation, and RSVP-TE protocol conformance. It provides positive as well as negative test cases against the RFCs that specify these standards. Negative tests help validate device response to **"killer** packets."

Test suite performs its tests as a dialog: it sends packets to the router being tested, receives the packets sent in response, and then analyzes the response to determine the next action to take. This allows it to test complicated situations or reactions in a much more intelligent and flexible way than can be done by simple packet generation and capture devices.

When the MPLS module tests are complete, we run a set of system tests to determine the functionality of the entire system. This includes the Network Services Module **(NSM)** and the protocol modules for RSVP-TE. This process validates the **communication,** configuration, and functionality for each of the protocol modules and the routing services module. This process includes route conversion, command line interface functionality, **management,** logging, and event handling.

5.1. Test cases

After setting up the hardware and software, we execute the desired test cases. The test cases are grouped into the following categories

(1) Ingress **LER** Test Cases; Egress LER Test Cases; (2) Intermediate LSR Test Cases; (3) Ingress LER Error Condition Test; (4) Egress LER Error Condition Test Cases and (5) Intermediate LSR Error Condition Test

All of the test cases test the **IUT's** ability to send the appropriate message to Tester connected to **IUT** in response to an event.

Conformance test suite is run on the emulator on 3 and 4 node topologies. Tests are done on LER (ingress and Egress) and intermediate LSRs.

6. IMPLEMENTATION

The MPLS RSVP-TE Emulator designed and implemented is successful in emulating MPLS networks. The emulated network can carry any network protocol traffic between the internal nodes. Some of the applications like SSH(Secure Shell), Secure copy **(SCP)** etc., **have** been tested. Some of **the** MPLS features like Traffic Engineering and VPNs are being tested, we have tried with 3-node and 4-node topologies.

7. EXPERIMENTS

We emulated MPLS on different topologies(see Figure 4). LSPs are created between ingress and egress nodes. Tunnels are created for the LSPs, and all the protocols are mapped to these tunnels. The following tables (see Table 1,2 and 3) are the forwarding label base created at different **UMLs** for the topologies emulated. For a given FEC on an LSP the **OutLabel** *PEEK* imply that LSR is the egress node.

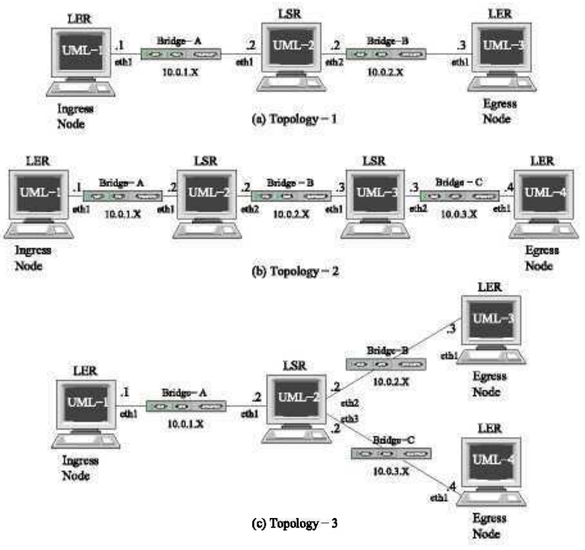


Fig. 4. Topologies Emulated

Table 1. for 3 node MPLS FIB

UML,	FEC	In-lbl	In-intf	Out-lbl	Out-intf
1	10.0.2.3			11680	ethl
1	10.0.1.1	11681	ethl	PEEK	
2	10.0.2.3	11680	ethl	21680	eth2
2	10.0.1.1	21681	eth2	11681	ethl
3	10.0.2.3	21680	ethl	PEEK	
3	10.0.1.1			21681	ethl

Table 2. for Linear 4 node MPLS FIB

UML,	FEC	In-lbl	In-intf	Out-lbl	Out-intf
1	10.0.2.3			11680	ethl
1	10.0.1.1	11681	ethl	PEEK	
2	10.0.2.3	11680	ethl	21680	eth2
2	10.0.3.4			21681	eth2
2	10.0.1.1	21682	eth2	11681	ethl
2	10.0.2.2	21683	eth2	PEEK	
3	10.0.2.3	21680	ethl	PEEK	
3	10.0.3.4	21681	ethl	31680	eth2
3	10.0.1.1			21682	ethl
3	10.0.2.2	31681	ethl	21683	ethl
4	10.0.3.4	31680	ethl	PEEK	
4	10.0.2.2			31681	ethl

Table 3. for Star 4 node MPLS FIB

UML,	FEC	In-lbl	In-intf	Out-lbl	Out-intf
1	10.0.3.4			11680	ethl
1	10.0.1.1	11681	eth1	PEEK	
2	10.0.3.4	11680	eth1	31680	eth3
2	10.0.3.4	21680	eth2	31681	eth3
2	10.0.1.1	31682	eth3	11681	ethl
2	10.0.2.3	31683	eth3	21681	eth2
3	10.0.3.4			21680	ethl
3	10.0.2.3	21681	ethl	PEEK	
4	10.0.3.4	31680	ethl	PEEK	
4	10.0.3.4	31681	ethl	PEEK	
4	10.0.1.1			31682	ethl
4	10.0.2.3			31683	ethl

8. CONCLUSION

In this paper, we have described an architecture of **UML** based MPLS Emulator. The emulator can be used as a test-environment for testing control protocol implementations. The APIs also enable MPLS emulator 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.

9. REFERENCES

[1] **E.Rosen,** A. **Viswanathan, R. Callon,** et **al., "Multi-**protocol Label Switching **Architecture,"***IETF Internet* ***Draft,*** Aug. 1999.

[2] Dike, **Jeff,** "A user-mode port of the Linux **kernel,"** ***In Proceedings of*** *the* ***4***th *Annual Linux* ***Showcase*** *& Conference,* Atlanta, page 63, **Atlanta, GA,** 2000.

[3] B. White, J. **Lepreau,** L. **Stoller, R. Ricci,** S. **Guruprasad, M. Newbold, M. Hibler,** C. Barb, and A. **Joglekar,** "An Integrated Experimental Environment for Distributed Systems and Networks," *Proceedings* ***of****Boston,* MA, Dec. 2002, pp. **255–270.**

[4] **Abhijit Gadgil** and **Abhay Karandikar, "LiME:** A Linux based MPLS Emulator," ***Proc. of NCC****2002,* 2002.

[5] M. **Allman,** A. **Caldwell,** and S. **Ostermann,** "ONE: The Ohio Network Emulator," *Technical Report TR-* ***1997*****, 1997.**

[6] U. **Black, "MPLS** and Label Switching Networks," *Prentice Hall Series in Advanced Communications Technologies,* Prentice Hall **PTR, 2001.**

[7] The User-mode Linux Kernel Home Page. **http://user-mode-linux. sourceforge.net/**

[8] **IST Teqila** Project Home Page, **http://ist-tequila.org**

[9] **"Nettest** MPLS rnterEMULATOR", **http://www.nettest.com/**