Overview of MPLS Network Simulator: Design and Implementation

Gaeil Ahn and Woojik Chun

Department of Computer Engineering, Chungnam National University, Korea
{fog1, chun}@ce.cnu.ac.kr

Table of Contents
1. Introduction
2. Architecture of MPLS Node
3. APIs for LDP and CR-LDP
4. Creating a MPLS Network with MNS
   1. Example 1: Simulating a basic MPLS functions
   2. Experiment Results of Example 1
5. Conclusion

1. INTRODUCTION

The MPLS network simulator (MNS) has been implemented by extending Network Simulator (NS). MNS has been implemented on Sun Unix system extending ns-2.1b5 program, NS version 2.1. To get the source program, example files, manual, and install guide of MNS click here.

- Download of MNS program executed on ns-2.1b5 program

The primary purpose of this work is to develop a simulator that enables to simulate various MPLS applications without constructing a real MPLS network.

The implementation scope of MPLS simulator is as follows:
- MPLS Packet switching -- label swapping operation, TTL decrement, and penultimate hop popping
- LDP -- handling LDP messages (Request, Mapping, Withdraw, Release, and Notification)
- CR-LDP -- handling CR-LDP messages (Request and Mapping)

The capability of MPLS simulator related to setting up LSP is as follows:
- In LSP Trigger Strategy -- support control-driven and data-driven trigger.
- In Label Allocation and Distribution Scheme -- support only downstream scheme in control-driven trigger, and both upstream and downstream-on-demand scheme in data-driven trigger.
- In Label Distribution Control Mode -- support only independent mode in control-driven trigger, and both independent and ordered mode in data-driven trigger.
- In Label Retention Mode -- support only conservative mode.
- ER-LSP based on CR-LDP -- established based on the path information pre-defined by user.
- Flow Aggregation -- aggregate fine flows into a coarse flow

2. ARCHITECTURE OF MPLS NODE

MNS is extended from NS that is an IP based simulator. In NS, a node consists of agents and classifiers. An agent is the sender/receiver object of protocol, and a classifier is the object that is responsible for the packet classification required for forwarding packets to the next node. For the purpose of making a new MPLS node from an IP node, 'MPLS Classifier' and 'LDP agent' are inserted into the IP node.

Figure 1 shows the architecture of a MPLS node in the MPLS simulator. 'Node entry' returns an entry point for the node. This is the first element that will handle packets arriving at the node. The Node instance variable, 'entry_.' , stores the reference to this element. Since multicasting is not supported the variable must be refer to the 'MPLS Classifier', which will first classify the incoming packet into labeled and unlabeled one. For unlabeled packet the packet is treated in conventional way. For labeled packet the 'MPLS Classifier' is responsible for label swapping and packet switching. The 'MPLS Classifier' instance variable, 'classifier_.' , stores the reference to another node or 'Addr Classifier'. 'Addr Classifier' is responsible for packet forwarding based on IP's destination address and 'Port Classifier' is responsible for agent selection.

On receiving a packet, a MPLS node operates as follows:
1. 'MPLS Classifier' determines whether the received packet is labeled or unlabeled. In case of a labeled packet, 'MPLS Classifier' executes L2 switching that sends it directly to the next node after label swapping operation. If it is an unlabeled packet but a LSP for the packet is prepared, 'MPLS Classifier' executes L2 switching as labeled packet forwarding. Otherwise, 'MPLS Classifier' sends it to 'Addr Classifier'.

-1-
2. 'Addr Classifier' executes L3 forwarding by examining the packet's destination address.
3. If the next hop of the packet is itself, the packet is sent to 'Port Classifier'.

A packet arrived

Figure 1: Architecture of MPLS Node

A MPLS node has three tables to manage information related to LSP, Partial Forwarding Table (PFT), Label Information Base(LIB), and Explicit Routing information Base (ERB). PFT is a subset of Forwarding Table and consists of FEC, PHB (Per-Hop-Behavior), and LIBptr. LIB table has information for LSP, and ERB table has information for ER-LSP. Figure 2 shows the structure of these tables and the simple algorithm for forwarding packets. The LIBptr in each table is a pointer to point to a LIB entry.

Figure 2: Structure of tables for MPLS packet switching

The lookup of PFT/LIB table is initiated when a MPLS node received a labeled/unlabeled packet. In the case of an unlabeled packet, the MPLS node searches PFT table for an entry with the packet's FEC (that is, packet's destination address). If the LIBptr of the found PFT entry indicates NULL, then the MPLS node forwards the packet by using L3 forwarding scheme. Otherwise, the MPLS node performs a label push operation for the packet. That is, it pushes into the packet the outgoing-label of the LIB entry pointed by a LIBptr of the PFT entry. In succession, there may be a label stack operation for the packet that a label push operation is repeated until the LIBptr of the LIB entry indicates NULL. After all label operation is finished, the packet is forwarded directly to a next node indicated by the outgoing-interface of the LIB entry.

In the case of a labeled packet, the MPLS node easily identify the LIB entry for it by using the label inserted in it as index into LIB table. Then, the MPLS node performs a label swap operation that replaces the packet's label with an outgoing-label of the LIB entry. If the outgoing-label is a null label, which means penultimate hop popping, the MPLS node performs a label pop operation instead of a label swap operation. And then, the MPLS node performs the label stack operation for the packet in case that the LIBptr of the LIB entry is not NULL. Finally, the packet is forwarded directly to a next node indicated by the outgoing-interface of the LIB entry.

The ERB table is used to keep only the information for ER-LSP. So, it doesn't participate in packet forwarding. If it is needed to map a flow into an previously established ER-LSP, a new entry which has the same LIBptr as that of its ERB entry should be inserted into PFT table.

3. APIs for LDP and CR-LDP

API Invocation

send-request-msg
get-request-msg
get-cr-request-msg
get-mapping-msg
get-cr-mapping-msg
get-notification-msg
get-withdraw-msg
get-release-msg

API Invocation

ldp-trigger-by-control
ldp-trigger-by-data
ldp-trigger-by-explicit-route
ldp-trigger-by-withdraw
ldp-trigger-by-release

API Invocation
**Figure 3: API Invocation for handling LDP API for Creating MPLS network**

When a LDP agent receives a LDP message, it should handle the message, select a next node, create a new LDP message, and send it to an agent attached to the next node. A sequence of the API invocation for this is described as Figure 3. When a LDP agent receives a LDP message, its API get-*msg is called. After handling the message, the LDP agent calls an API ldp-trigger-by-*, which belongs to its MPLS node, in order to select a next node to receive a LDP message. Once a next node is determined, a MPLS node calls an API send-*,msg of its agent corresponding to a agent of the next node in order to create a new LDP message and send it to the next node.

### 4. Creating a MPLS Network with MNS

The followings are APIs defined for creating a MPLS network:

- **MPLSNode** -- create a new MPLS node
- **configure-ldp-on-all-mpls-nodes** -- attach LDP agents to all MPLS node.
- **enable-control-driven** -- let all LSRs operate as control-driven trigger.
- **enable-traffic-driven** -- let LSRs operate as traffic-driven trigger.
- **enable-on-demand** -- let LSRs operate as on-demand label allocation mode
- **enable-ordered-control** -- let LSRs operate as ordered mode
- **make-explicit-route** -- establish an ER-LSP
- **flow-erlsp-binding** -- map a flow onto an established ER-LSP.
- **flow-aggregation** -- aggregate fine flows into a coarse flow
- **trace-mpls** -- trace MPLS packets
- **trace-ldp** -- trace LDP packets

**Figure 4: Example of MPLS Packet Trace**

When API 'trace-mpls' is used, an example of trace result might appear as Figure 4. First field indicates the simulated time (in seconds) at which each event occurred. The next field indicates the node address that processes the packet. The next two fields indicate the packet's source and destination node addresses. The next field indicates whether the received packet is unlabeled(U) or labeled(L). The next field is an incoming-label value. The next field represents a label operation such as Push, Pop, and Swap. The subsequent two fields indicate the packet's out-going interface and out-going label. The last two fields indicate the shim header's TTL and size.

### 4.1. Example 1: Simulating a basic MPLS functions

Figure 5 shows the experiment environment for simulating basic MPLS functions.

**Figure 5: an Example of MPLS Network**

In Figure 5, Node0, Node1, Node9, and Node10 are IP nodes, and the others are MPLS nodes. Src0 agent attached to Node0 sends packets toward Dst0 agent attached to Node9. Src1 agent attached to Node1 sends packets toward Dst1 agent attached to Node10. Under a packet forwarding scheme based on shortest path, packets from Src0 are delivered along LSR 2-5-6-7, and packets from Src1 are delivered along LSR 2-3-4-8.

The following is the code written in Tcl for constructing the MPLS network described in Figure 5.

First of all, We have to create a simulator object. This is done with the command as follows:

- set ns [new Simulator]

Four IP node objects and seven MPLS node objects are created as follows:

- set Node0 [ns node]
- set Node1 [ns node]
- set LSR2 [ns MPLSNode]
- set LSR3 [ns MPLSNode]
- set LSR4 [ns MPLSNode]
- set LSR5 [ns MPLSNode]
- set LSR6 [ns MPLSNode]
- set LSR7 [ns MPLSNode]
We could choose data-driven trigger scheme instead of control-driven. The command "$ns enable-data-driven" let all MPLS nodes operate as data-driven trigger. In case that data-driven trigger scheme is selected, we can use the following options:

- $ns enable-on-demand
- $ns enable-ordered-control

The next lines are the code for creating CBR agents as traffic sources and attach them to the nodes Node0 and Node1, and then Null agents and attach them to node Node9 and Node10.

- set Src0 [new Agent/CBR]
- $ns attach-agent $Node0 $Src0
- $Src0 set packetSize _ 500
- $Src0 set interval_ 0.010
- set Src1 [new Agent/CBR]
- $ns attach-agent $Node1 $Src1
- $Src1 set packetSize _ 500
- $Src1 set interval_ 0.010
- set Dst0 [new Agent/Null]
- $ns attach-agent $Node9 $Dst0
- set Dst1 [new Agent/Null]
- $ns attach-agent $Node10 $Dst1

The Two CRB agents have to be connected to the Null agents as follows:

- $ns connect $Src0 $Dst0
- $ns connect $Src1 $Dst1

The followings are a code for the event scheduling in this simulation. First, at 0.1 seconds, Src0 sends packets to Dst0, and Src1 sends packets to Dst1. At 0.2 seconds, LSR7, which is egress LSR for FEC 9, terminates a LSP for FEC 9, and LSR8, which is egress LSR for FEC 10, terminates a LSP for FEC 10. At 0.3 seconds, flows of FEC 9 and FEC 10 are aggregated into a flow of FEC 6. At 0.5 seconds, a LSP for FEC 6 is terminated, and Src1 agent stop generating packet at 0.7 seconds.

Subsequently, at 0.7 seconds, an ER-LSP of which LSPID is 3000 is established between LSR2 and LSR7 through LSR 5-4-8-6-7. At 0.9 seconds, a flow of FEC 9 is bound to the established ER-LSP. At 1.1 seconds, the ER-LSP is terminated with LDP Release Message.
And then, at 1.2 seconds, an ER-LSP Tunnel of which LSPID is 3500 is established between LSR4 and LSR8 through LSR5 and LSR6. An ER-LSP of which LSPID is 3600 is also established between LSR2 and LSR7 through LSR 2-3-4-3500-7 at 1.4 seconds. 3500 in the explicit route means LSPID, which is used to identify the tunnel ingress point as a next hop. This allows for stacking new ER-LSP (that is, LSPID 3600) within an already established LSP Tunnel (that is, LSPID 3500). At 1.6 seconds, the flow of FEC 9 is bound to the established ER-LSP. Finally, at 2.0 seconds Src0 stops generating packet.

By using the commands, pft-dump, erb-dump, and libdump, the state of the established LSP is dumped.

| $ns at 0.1 | "$Src0 start"  |
| $ns at 0.1 | "$Src1 start"  |
| $ns at 0.2 | "$LSR7 ldp-trigger-by-withdraw 9 -1" |
| $ns at 0.2 | "$LSR8 ldp-trigger-by-withdraw 10 -1" |
| $ns at 0.3 | "$LSR2 flow-aggregation 9 -1 6 -1" |
| $ns at 0.3 | "$LSR2 flow-aggregation 10 -1 6 -1" |
| $ns at 0.3 | "$LSR6 ldp-trigger-by-withdraw 6 -1" |
| $ns at 0.3 | "$LSR6 ldp-trigger-by-withdraw 5 -1" |
| $ns at 0.7 | "$LSR2 make-explicit-route 7 5_4_8_6_7 3000 -1" |
| $ns at 0.9 | "$LSR2 flow-erlsp-install 9 -1 3000" |
| $ns at 0.9 | "$LSR2 flow-erlsp-install 10 -1 3000" |
| $ns at 1.1 | "$LSR2 ldp-trigger-by-release 7 3000" |
| $ns at 1.2 | "$LSR4 make-explicit-route 8 4_5_6_8 3500 -1" |
| $ns at 1.4 | "$LSR2 make-explicit-route 7 2_3_4_3500 7 3600 -1" |
| $ns at 1.6 | "$LSR2 flow-erlsp-install 9 -1 3600" |
| $ns at 2.0 | "$Src0 stop" |
| $ns at 2.0 | "$LSR2 pft-dump" |
| $ns at 2.0 | "$LSR2 erb-dump" |
| $ns at 2.0 | "$LSR2 lib-dump" |
| $ns at 2.0 | "$LSR3 pft-dump" |
| $ns at 2.0 | "$LSR3 erb-dump" |
| $ns at 2.0 | "$LSR3 lib-dump" |
| $ns at 2.0 | "$LSR4 pft-dump" |
| $ns at 2.0 | "$LSR4 erb-dump" |
| $ns at 2.0 | "$LSR4 lib-dump" |
| $ns at 2.0 | "$LSR5 pft-dump" |
| $ns at 2.0 | "$LSR5 erb-dump" |

4.2. Experiment Results of Example 1

Figure 6 are pictures shown when the simulation result generated by using MPLS simulator is executed with NAM.

Figure 6-a shows the initial simulated network. Figure 6-b shows a snapshot at 0.01 seconds that LDP Mapping Message is used to distribute label based on control-driven trigger. As the result, every LSP in the MPLS network is established. Figure 6-c shows a snapshot at 0.16 seconds that packets are forwarded along a established LSP. For example, in case of a packet sent by Src0, there is label push operation in LSR2, label swap operation in LSR5, and label pop operation in LSR6 that is a penultimate hop. Figure 6-d shows a snapshot at 0.20 seconds that a LSP for FEC 9 and a LSP for FEC 10 are terminated with LDP withdraw message. Figure 6-e shows a snapshot at 0.34 seconds after flows of FEC 9 and FEC 10 were aggregated into a flow of FEC 6.

Subsequently, Figure 6-f shows a snapshot at 0.70 seconds that CR-LDP Request Message initiated by LSR2 is delivered along LSR 5-4-8-6 in order to create an ER-LSP between LSR2 and LSR7. Figure 6-g shows a snapshot at 0.76 seconds that CR-LDP Mapping Message is sent by LSR7 as the response for the LDP Request Message initiated by LSR2. Figure 6-h shows a snapshot at 0.97 seconds that packets are forwarded along the ER-LSP created through step shown in Figure 6-f and Figure 6-g.

Figure 6-i shows a snapshot at 1.69 seconds that traffic is forwarded along an established ER-LSP, which includes an already established LSP Tunnel. A step to establish the ER-LSP and the LSP Tunnel is not shown in Figure 6 because it is similar to that to establish an
ER-LSP shown in Figure 6-f and Figure 6-g. LSR4 is a Tunnel Ingress point and LSR8 is a Tunnel Egress point. When a packet is forwarded along the ER-LSP, the trace result of a MPLS packet appears as Figure 7. There is a label push operation in LSR2, a label swap operation in LSR3 and LSR4, a label push operation in LSR4, a label swap operation in LSR5, and a label pop operation in LSR6 and LSR8.

(a) At 0.0 seconds: Initiation

(b) At 0.01 seconds: Label Distribution based on control-driven trigger

(c) At 0.16 seconds: L2 packet switching

(d) At 0.20 seconds: Label Withdrawal for FEC 9 and FEC 10

(e) At 0.34 seconds: Flow Aggregation of FEC 9 and FEC 10 into FEC 6

(f) At 0.70 seconds: CR-LDP Request Message for creating an ER-LSP

(g) At 0.76 seconds: CR-LDP Mapping Message for creating an ER-LSP

(h) At 0.97 seconds: Packet Switching based on the established ER-LSP
(i) At 1.69 seconds: Packet Switching based on the established LSP Tunnel

**Figure 6: Simulation Results viewed with NAM**

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Type</th>
<th>Source</th>
<th>Action</th>
<th>Index</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.634000</td>
<td>2(0-&gt;9):  U -1 Push(ingress)</td>
<td>3</td>
<td>11</td>
<td>32</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.648000</td>
<td>3(0-&gt;9):  L 11 Swap</td>
<td>4</td>
<td>12</td>
<td>31</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.662000</td>
<td>4(0-&gt;9):  L 12 Swap</td>
<td>8</td>
<td>11</td>
<td>30</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.662000</td>
<td>4(0-&gt;9):  L 12 Push(tunnel)</td>
<td>5</td>
<td>12</td>
<td>32</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.676000</td>
<td>5(0-&gt;9):  L 12 Swap</td>
<td>6</td>
<td>12</td>
<td>31</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.690000</td>
<td>6(0-&gt;9):  L 12 Pop(penultimate)</td>
<td>8</td>
<td>0</td>
<td>30</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.704000</td>
<td>8(0-&gt;9):  L 11 Pop(penultimate)</td>
<td>7</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.718000</td>
<td>7(0-&gt;9):  U -1 L3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7: MPLS Packet Trace for Figure 6-i**

5. **CONCLUSION**

I have introduced the MPLS Network Simulator that support MPLS functionality extending ns simulator. I have given a example of the use of the MPLS Network Simulator.