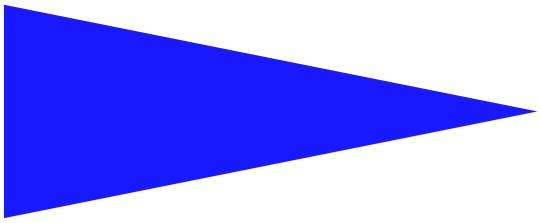
I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1493



A SIMULATOR FOR MULTICAST ROUTING OVER AN MPLS NETWORK

ALI BOUDANI, CHADI JAWHAR, BERNARD COUSIN AND MAHMOUD DOUGHAN

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

A Simulator for Multicast Routing over an MPLS Network

Ali Boudani, Chadi Jawhar, Bernard Cousin and Mahmoud Doughan*

Thème 1 — Réseaux et systèmes
Projet Armor

Publication interne n ° 1493 — October 2002 — 29 pages

Abstract: Multicast and MPLS are two complementary technologies. Merging these two technologies where multicast trees are constructed over MPLS networks will enhance performance and present an efficient solution for multicast scalability and control overhead problems. In this paper** , we present a simulator for multicast routing over an MPLS network. We briefly discuss MPLS, multicast, benefits resulting from merging the MPLS and the multicast technologies and, existing MPLS network simulator. We present finally our simulator for multicast routing over MPLS network where we choose PIM-SM (source specific tree) as the multicast routing protocol. Our basic idea is to preserve the existing code for unicast transmission simulation using the MPLS networks simulator (MNS). Unicast label distribution, LSP construction and L2 switching still functioning the same.

A simulator for multicast routing over MPLS network is an original idea since this kind of simulator never existed before and it will help researchers to simulate and evaluate their MPLS multicast related techniques.

Key-words: MPLS, PIM-SM, NS, Multicast

(Résumé : tsvp)

* { Ali.Boudani}{Bernard.Cousin}@irisa.fr,{Chadi.Jawhar}{mdoughan}@ul.edu.lb
** This work has been supported by the franco-lebanese program CEDRE

Centre National de la Recherche Scientifique (UPRESSA 6074) Université de Rennes 1 – Insa de Rennes
Institut National de Recherche en Informatique et en Automatique – unité de recherche de Rennes

Un simulateur pour le routage multicast dans un réseau MPLS

Résumé : Multicast et MPLS sont deux technologies complémentaires. La combinaison de ces deux technologies où des arbres multicast sont construits dans des réseaux MPLS présente une solution efficace pour la scalabilité multicast. Dans cet article, nous présentons un simulateur pour le routage multicast dans un réseau MPLS. Nous discutons brièvement de MPLS, du multicast, des avantages résultants de la combinaison de MPLS et du multicast et, du simulateur existant de MPLS (MNS). Nous présentons finalement notre simulateur pour le routage multicast dans un réseau MPLS et nous choisissons PIM-SM (arbre spécifique à une source) comme protocole de routage multicast. Notre idée fondamentale est de préserver le code existant pour la simulation de transmission unicast du simulateur de réseaux MPLS (MNS). La distribution de label unicast, la construction des LSP et la commutation de label de niveau 2 ont le même fonctionnement.

Un simulateur pour le routage multicast dans un réseau MPLS est une idée originale car ce genre de simulateur n'a jamais existé avant et ce simulateur peut aider les chercheurs à simuler et évaluer leurs techniques combinant le multicast et MPLS.

Mots clés : MPLS, PIM-SM, NS, Multicast

Contents

1 Introduction

Several evolving applications like WWW, video/audio on-demand services, and teleconferencing consume a large amount of network bandwidth. Multicasting is a useful operation for supporting such applications. Using the multicast services, data can be sent from a source to several destinations by sharing the link bandwidth.

But multicast suffers from the scalability problem. Indeed, a multicast router should keep forwarding state for every multicast tree passing through it. The number of forwarding states grows with the number of groups.

Besides, Multi-protocol label switching (MPLS) [1] has emerged as an elegant solution to meet the bandwidth-management and service requirements for next generation Internet protocol (IP) based backbone networks. We think that Multicast and MPLS are two complementary technologies, and merging these two technologies, where multicast trees are constructed in MPLS networks will enhance performance and present an efficient solution for multicast scalability and control overhead problems.

This paper proposes a simulator for multicast routing over an MPLS network by extending MPLS Network Simulator (MNS) [2].

NS [3] is a network simulator intended for studying the dynamic behaviour of flows and congestion schemes in a network. The simulator takes as input a scenario, which is a description of network topology, protocols, workload and control parameters. The simulation results from NS may be shown with Graphic User Interface (GUI) that is called Network Animation (NAM) [4]. NAM is an animation tool for viewing network simulation traces and real world packet traces. It supports topology layout, packet level animation, and various data inspection tools.

MPLS is implemented in NS (the MPLS network simulator (MNS) [2]) with all its features, from the label distribution to the layer two switching data transmission. Besides, many multicast routing protocols (PIM-SM [5], PIM-DM [6], etc.) are also implemented in NS. In the current NS implementation, every MPLS node or multicast node has its own specific classifier to process respectively labeled packets or multicast packets. Therefore, when a node is defined as an MPLS-capable node, it cannot be configured also as multicast-capable node. As a result, modifications to the MPLS simulation code in NS are needed so that an MPLS-capable node can understand and manage multicast traffic.

The remainder of this paper is organized as follows. In this Section, we present multicast, MPLS, related work and PIM-SM implementation in MPLS. Section 2 descibes how MPLS is implemented in NS. We explain how LDP protocol [7] is defined in NS, and how the L2 switching is executed. Section 3 describes the commands needed for simulation examples. We finally present in Section 4 our implementation of multicast routing over an MPLS network with PIM-SM (source specific mode only) as the multicast routing protocol. Section 5 is a summary followed by appendixes and a list of references.

1.1 Multicast

Multicast has become increasingly important with the emergence of network-based applications such as IP telephony, video conferencing, distributed interactive simulation and software upgrading.

Using multicast service, a single transmission is needed for sending a packet to n destinations, while n independent transmissions would be required using unicast service. A multicast routing protocol should be simple to implement, scalable, robust, use minimal network

overhead, consume minimal memory resources, and inter-operate with other multicast routing protocols [8].

Many multicast protocols have been proposed and are in use today on the Internet. They include (but not limited to) DVMRP, MOSPF, PIM-SM, PIM-DM, CBT, BGMP (see [8] for more details about these protocols). The differences between these protocols lies mainly in the type of multicast routing trees they build. DVMRP, MOSPF, and PIM-DM build multicast spanning trees that use shortest paths from every source to any destination. PIM-SM, CBT build spanning trees that are shortest path from a known central core, also called rendez-vous point (RP), where all sources in the session share the same spanning tree. PIM-SM is the most widely implemented protocol. It is a complicated protocol that at times builds source-rooted shortest path trees. An IP group address range has been designated for source-specific multicast (SSM [9]) applications and protocols and should support source-only trees (source specific mode), precluding the requirement of an RP and a shared tree.

1.2 Multi-Protocol Label Switching

MPLS is a versatile solution to address the problems faced by present day networks (speed, scalability, quality-of-service (QoS) management, and traffic engineering). MPLS is Multi-protocol because it can be applied with any layer 3 network protocol, although almost all of the interest is in using it with IP traffic. MPLS is about gluing connectionless IP to connection-oriented networks. It is something between Layer 2 and Layer 3 that makes them fit better. MPLS is an advanced forwarding scheme that extend routing with respect to packet forwarding and path controlling. Packets are classified easily at domain entry, and rerouted faster (in the case of link failures) and explicite routes are easy to construct.

An MPLS domain is a contiguous set of routers which operate MPLS routing and forwarding and which are also in one routing or administrative domain [1]. An MPLS capable router is called LSR (label switching router).

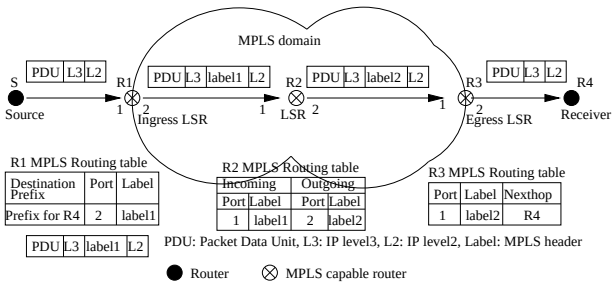


Figure 1: MPLS forwarding scheme

At the ingress LSR of an MPLS domain, IP packets are classified and routed in FECs (forwarding equivalence class) based on a combination of the information carried in the IP header of these packets and local routing information maintained by the LSR. Once a packet is assigned to a FEC, no further header analysis is done by subsequent routers in the same MPLS domain. An MPLS header, called label, is inserted for each packet within an MPLS domain, an LSR will use the label as the index to look up the forwarding table of the LSR. The packet is processed as specified by the forwarding table entry. The incoming label is replaced

by an outgoing label, and the packet is switched toward the next LSR. Before a packet leaves an MPLS domain, its MPLS header is removed [11]. The paths between the ingress LSRs and egress LSRs are called label-switched paths (LSPs). MPLS uses some signaling protocol such as Resource Reservation Protocol (RSVP) [10] or Label Distribution Protocol (LDP) [7] to set up LSPs. The forwarding process is shown in Fig.1.

MPLS shows a number of advantages over conventional network layer forwarding:

- MPLS forwarding can be done by switches which are only capable of doing label lookup and replacement, but not necessary being capable of analyzing the network layer headers, or analyzing the network layer headers at adequate speed.
- Since a packet is assigned to a FEC when it enters the network, the ingress router may use, in determining the assignment, any information it has about the packet, even if that information cannot be gleaned from the network layer header. For example, packets arriving on different ports may be assigned to different FECs. Conventional forwarding, on the other hand, can only consider information in the packet header. The considerations that determine how a packet is assigned to a FEC can become ever more and more complicated without any impact at all on LSR that merely forward labeled packets.
- Sometimes it is desirable to force a packet to follow a particular route which is explicitly chosen at or before the time the packet enters the network, rather than being chosen by the normal dynamic routing algorithm as the packet travels through the network. This may be done as a matter of policy, or to support traffic engineering. In conventional forwarding, this requires the packet to carry an encoding of its route along with it ("source routing"). In MPLS, a label can be used to represent the route, so that the identity of the explicit route need not be carried with the packet.

From this point of view, and focusing on the advantages of the layer two switching protocol, Multicasting over MPLS networks can benefit from the multicast reduce of traffic on one hand, and MPLS flexibility, speed and quality of service on the other hand.

1.3 Related Work

A framework for MPLS multicast traffic engineering proposed by Ooms et al [12] gives an overview about the application of MPLS techniques to IP multicast. Another study about MPLS and multicast proposed by Farinacci et al. [13] explains how to use PIM to distribute MPLS labels for multicast routes. A piggy-backing method is suggested to assign and distribute labels for multicast traffic for sparse-mode trees. Another proposal is aggregated multicast [14]. The key idea of aggregated multicast is that, instead of constructing a tree for each individual multicast session in the core network, one can have multiple multicast sessions sharing a single aggregated tree to reduce multicast state and, correspondingly, tree maintenance overhead at network core.

A new approach to construct multicast trees in MPLS networks [15] was proposed recently. In that approach, MPLS LSPs are used between multicast tree branching node routers in order to reduce forwarding states and enhance scalability. Only routers that are acting as multicast tree branching nodes for a group need to keep forwarding states for that group. All other non-branching node routers simply forward data packets over traffic engineered unicast routes using MPLS LSPs.

1.4 PIM-SM Implementation in MPLS

The implementation of the PIM-SM protocol over MPLS networks is done in a very simple manner. The idea is that in the branching routers, instead of mapping the <incoming Label, incoming Interface> to one <outgoing Label, outgoing Interface>, the mapping is done to several outgoing interfaces according to the distribution of the group members. When a data packet arrives, instead of doing only one label switching, the data packet is replicated, and for each copy a label switching is done. These copies are transmitted then to the convenient outgoing interfaces.

Fig.2 shows 2 group members (R6 and R9) that want to join the (S1, G1) session where S1 is the source at R1 and G1 is the group address. When R6 joins first the session, labeling will be similar to the one used in unicast with only one difference: the allocated labels are associated to an (S1, G1) entry. The labeling is done from the downstream member up to the source. When R9 joins the same session, the labeling will be done in the same manner from the member toward the source. When the labeling message reaches a tree node (a node with an (S1, G1) entry), it will not forwarded anymore and the node becomes a branching node. A branching node contains one <incoming label, incoming interface> mapped to more than one <outgoing label, outgoing interface>. The upstream labeling is done from the new member R9 until reaching the router R3. The router R3 is already a tree node for the (S1, G1) session, and therefore no further upstream labeling is done. The R3 becomes then a branching node for the (S1, G1) session.

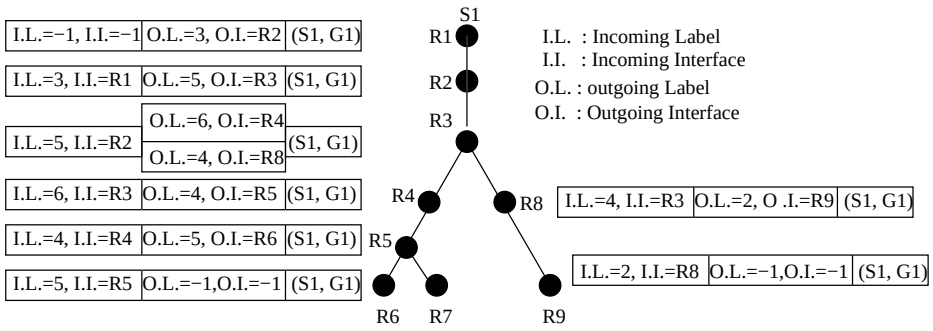


Figure 2: MPLS multicast routing table entries for (S1, G1) session

Data forwarding is done similarly as in unicast mode. When the source S1 wishes to transmit data to the group G1, it will lookup in its information base, find the label corresponding to the (S1, G1) session, and label the data packets and transmit them toward the outgoing interface. All intermediate routers have only to switch the label, and forward the packets, exactly as in unicast. In case of a branching node, it replicates the packet as many times as there are outgoing labels associated with input label in its information base, and for each packet copy it will do the label swapping and transmit it on the respective outgoing interface. The pruning is done also from the pruned member up toward the source. At each node a label deallocation is done until reaching either the source or a branching node. If it reaches a branching node, only the corresponding <outgoing interface, outgoing label> is removed and the branching node will become a tree node.

2 How MPLS is Implemented in NS

This section describes the design and the implementation of the MPLS protocol in the network simulator NS. MNS [16] (MPLS network simulator), supports two main functions:

- LDP Label Distribution Protocol.
- MPLS Label Switching.

NS is an IP based simulator where a node consists of classifiers and agents. An agent is the sender/receiver object, while the classifier is the object that is responsible for classifying the arrived packets and then either forwarding them to the convenient nodes or delivering them to the local agent if the receiving node is the packet destination. Therefore, in order to construct an MPLS node, a new classifier, called the MPLS classifier, should be created in order to be able to classify the received packets, determines whether they are labeled or not, and treat them correspondingly. Also, a new agent, the LDP agent, must be also inserted in the IP node in order to distribute labels to other MPLS nodes and construct the LSP paths (see Fig. 3).

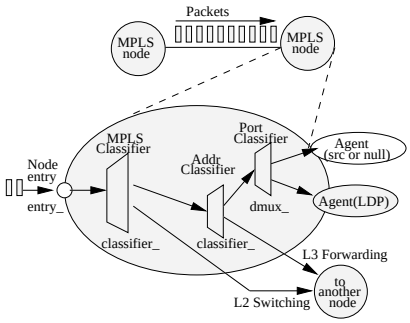


Figure 3: MPLS node architecture in NS

An MPLS node has three tables to manage the information related to the LSP and the label distribution; Partial Forwarding Table (PFT), Label Information Base (LIB), and Explicit Routing information Base (ERB). PFT table is a subset of forwarding table and consists of FEC, PHB (Per-Hop-Behavior), and LIBptr fields. LIB table has information for LSP, and ERB table has information for ER-LSP. Figure 4 shows the structure of these tables and gives indication about the forwarding process. The LIBptr in each table is a pointer to an LIB entry.

The LIB table is constructed and used to map the <Incoming Label, Incoming interface> to the <Outgoing Label, Outgoing interface>. It is then used when L2 operation is to be executed: when a labeled packet is received and a label swap is to be done or when an unlabeled packet is received and a label push is required.

The PFT table is used when an unlabeled packet arrives. The MPLS node will search this table for an entry where the FEC is the packet's destination address. The entry could either point to an entry in the LIB table to perform the convenient label push operation or point to NULL and as a result ordinary L3 forwarding is being done.

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

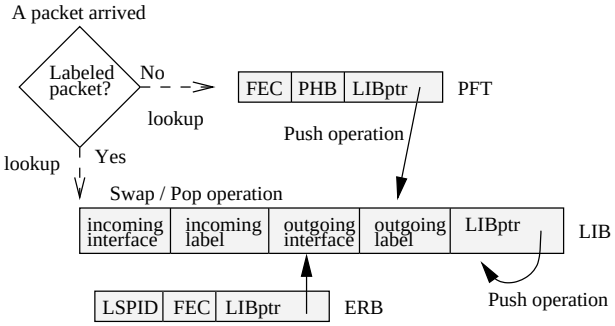


Figure 4: Structure of tables for MPLS packet switching

2.1 Label Distribution

In MNS, the distribution of labels and the construction of LSPs is done by exchanging LDP messages between the LDP agents of LSR nodes. MNS offers three modes of label distribution:

- Control Driven
- Data Driven
- Explicit Routing

Control driven mode relies on distributing LDP messages between all LDP agents even if there is no data to be transmitted. LSPs are constructed for each FEC and this is done by sending mapping messages from each LDP agent to all the others, containing the FEC along with the label that should be used later for the data transmission. At the end, all LIB tables of all MPLS nodes are filled and different LSPs are assigned for all FECs.

Data driven mode distributes LDP messages and constructs LSPs only for FECs which are the destinations of source agents which desire to transmit data. Therefore, when a node wishes to transmit data it sends a request message to the FEC. The first packets transmitted are forwarded as layer 3 packets until the LSP is constructed, then L2 switching can be done. When the FEC receives the request message, it sends a mapping message upstream toward the source and each router in the way receives a mapping message, handles it, creates a new LDP message and transmits it to the nexthop toward the source. In this way an LSP is constructed from the source toward the destination.

In the explicit routing labeling, LSPs are constructed in a simple way. The user needs to insert the successive nodes of the explicit route which data packets will follow. Mapping messages are distributed only along this path and then construct the LSP for this FEC.

2.2 Releasing and Withdrawing an LSP

Releasing an LSP is used when an explicit route has been set up and there is no need for it anymore. Also, another need of releasing an LSP is when a node, in data driven mode, need to temporary transmit data to a certain destination. So it must construct a temporary LSP which must be released after the transmission ends. A node will clear the entries corresponding to the FEC (and corresponding to the LSPid when deleting an explicit routed LSP) from the

information base tables, and will send a release message to the nexthop node which is on the corresponding outgoing interface. When this nexthop receives this release message containing the FEC and the LSPid, it will clear its entries and in turn sends a release message toward its own outgoing interface until reaching the FEC or a non LSR node. In this manner the LSP will be released and the deallocated labels are now free for future allocation.

Withdrawing an LSP is another method of LSP releasing. Executing this function will release the LSP set up toward a node corresponding to FEC and LSPid. The node will send a withdraw messages to all the previous hop nodes which are associated to the incoming interfaces corresponding to the (FEC, LSPid) and will erase their allocated labels inside their node's information base. When the upstream nodes receive the withdraw messages, they will free up their allocated outgoing labels and run the withdram function. In this way all the upstream labels and LSPs corresponding to the (FEC, LSPid) will be deleted.

2.3 MPLS Label Switching

When a packet arrives at a certain node, it is handled by the classifier (MPLS classifier) which classify it, process it, and forward it either to a local agent or to another node. Simply, the packet process is done as shown in the following algorithm:

- The ingress LSR may not have any label for this packet as it is the first occurrence of this packet type. It will lookup for the longest prefix match, find the the nexthop router and initiates a label request toward it. This label request will propagate through the network from the ingress LSR to the egress LSR. Each intermediate LSR will receive a label from its downstream LSR, install an entry in its LIB table, and then choose a new label and transmit it to its upstream LSR.
- The ingress LSR will insert the label and forward the packet to the nexthop LSR.
- Each intermediate LSR, will examine the label in the received packet, replace it with the outgoing label and forward it based on their LIB table.
- When the packet reaches the egress LER, it will remove the label because the packet is departing from an MPLS domain and forward it toward to the destination.

3 MPLS Simulation

This section describes the commands used in order to simulate an MPLS topolgy in our simulation followed by a simulation example. To simulate MPLS networks one should know the commands used to define an MPLS node, execute different modes of label distribution, release LSPs, trace MPLS and LDP packets, and finally use utility commands.

3.1 Topology Creation/Generation Commands

- Creation of an MPLS node: (Say LSR0)

```
$ns node-config -MPLS ON
set LSR0  [$ns node]
$ns node-config -MPLS OFF
```

or

```
set LSR0  [$ns MPLSnode]
```

- Configuration of LDP agents on all MPLS nodes: (say n nodes whose names are LSRi, i is an integer)

```
for {set i 0} {$i < n} {incr i} {
        set a LSR$i
        for {set j [expr $i+1]} {$j < 5} {incr j} {
                set b LSR$j
                eval $ns LDP-peer $$a $$b
        }
        set m [eval $$a get-module "MPLS"]
        $m enable-reroute "new"
}
```

or

```
$ns configure-ldp-on-all-mpls-nodes
```

3.2 Label distribution and LSP releasing commands

- Choice of a unique mode of label distribution for all MPLS nodes (note that by default the data driven mode is taken):

```
$ns enable-control-driven  or  classifier/Addr/MPLS set control_driven_ 1
$ns enable-data-driven     or  classifier/Addr/MPLS enable-data-driven
$ns enable-on-demand       or  classifier/Addr/MPLS enable-on-demand
$ns enable-ordered-control or  classifier/Addr/MPLS enable-ordered-control
```

- Choice of a distribution mode on a certain node (say MPLSnode):

```
[$MPLSnode get-module "MPLS"] enable-control-driven
[$MPLSnode get-module "MPLS"] enable-data-driven
[$MPLSnode get-module "MPLS"] enable-on-demand
[$MPLSnode get-module "MPLS"] enable-ordered-control
[$MPLSnode get-module "MPLS"] make-explicit-route fec er LSPid rc
[$MPLSnode get-module "MPLS"] ldp-trigger-by-release fec LSPid
```

3.3 Tracing commands

- Packet trace for an MPLS node (say MPLSnode): :

```
[$MPLSnode get-module "MPLS"] trace-mpls
[$MPLSnode get-module "MPLS"] trace-ldp
```

- Displaying information tables managed by an MPLS node (say MPLSnode):

```
[$MPLSnode get-module "MPLS"] pft-dump
[$MPLSnode get-module "MPLS"] lib-dump
[$MPLSnode get-module "MPLS"] erb-dump
```

3.4 Utility commands

Color settings for different types of LDP packets:

```
$ns ldp-request-color $color
$ns ldp-mapping-color $color
$ns ldp-withdraw-color $color
$ns ldp-release-color $color
$ns ldp-notification-color $color
```

3.5 Simulation Example

The topology example is shown in figure 5, where MPLS nodes are LSR1 to LSR7 forming an MPLS domain, and Node0, Node7, and Node8 are not MPLS-capable nodes. In this example we select the control mode on all the defined MPLS nodes as label distribution protocol.

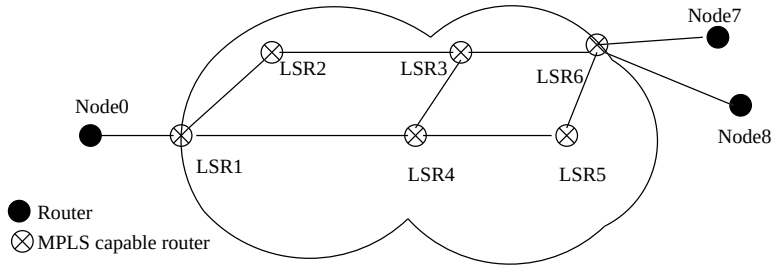


Figure 5: Structures of tables for MPLS packet switching

3.6 Simulation Results

The simulation results collected are mainly two types: the *nam* file showing graphically all packet transmissions between the created nodes, and the *trace* file which shows the trace of the MPLS, LDP and DV packets, and a display of the labeling tables at each node. We focus our interrest on the second type only. The trace of LDP packets (LDP packets means all the mapping, request, withdraw and release messages used for label distribution and LSP release) at node LSR1 is as follows:

```
0.074218 1: 2 (Mapping 1) 2 0 *_2  [-1 *]  [-1 * -1]
0.07421830807: <mapping-msg> 2 -> 1 : fec(2), label(0) 2
0.07421830810 1(1->4): U -1 L3 -1 -1 -1 0
```

Also, the MPLS packets could be traced using the trace-mpls command. This trace shows the push (L3 to L2), Swap (L2 to L2) and Pop (L2 to L3) label operations :

```
0.567850000000001 1(0->7): U -1 Push(ingress) 2 6 32 4
0.571600000000001 1(0->7): U -1 Push(ingress) 2 6 32 4
0.575350000000001 1(0->7): U -1 Push(ingress) 2 6 32 4
```

The information tables of LSR5 are shown in the next tables. These labels have been distributed based on the control mode that is chosen to be executed at LSR5 (ERB table is an empty table since there is no explicit routes defined).

```
     ___PFT dump___ [LSR: 5]
-------------------------------------------
    FEC       PHB     LIBptr  AltanativePath
     4         -1          0         -1
     0         -1          1         -1
     1         -1          2         -1
     6         -1          3         -1
     7         -1          4         -1
     8         -1          5         -1
     2         -1          6         -1
     3         -1          7         -1
    ___LIB dump___ [LSR: 5]
-------------------------------------------
     #     iIface     iLabel   oIface    oLabel    LIBptr
     0:      -1          1       4         0         -1
     1:      -1          2       4         1         -1
     2:      -1          3       4         2         -1
     3:      -1          4       6         0         -1
     4:      -1          5       6         0         -1
     5:      -1          6       6         0         -1
     6:      -1          7       4         5         -1
     7:      -1          8       6         1         -1
     ___ERB dump___ [LSR: 5]
-------------------------------------------
    FEC     LSPid      LIBptr
```

4 Implementing the simulator for multicast routing in MPLS networks

Multicasting over MPLS networks can benefit from the multicast reduction of traffic on one hand, and MPLS flexibility, speed and quality of service on the other hand. Many protocols have been proposed and are in use today on the Internet. The implementation of multicasting over MPLS networks must be done for each one of these multicast protocols.

PIM-SM is the most widely implemented protocol. It is a complicated protocol that at times builds source-rooted shortest path trees. An IP group address range has been designated for source specific multicast (SSM) applications and protocols and should support source oriented trees (source specific mode), precluding the requirement of an RP and a shared tree. This work focuses on the study of the PIM-SM protocol (source specific mode) over MPLS networks but it can be adapted to other protocols as well. This simulator is based on the piggybacking proposition [13] where a label is piggybacked by the join message in PIM-SM protocol.

MPLS code in NS does not work with multicast routing, particularly because (1) there is no label setup mechanism for multicast groups, (2) there is no multicast replicator to cooperate

with MPLS classifier, and (3) MPLS header contains pointers, which do not work with multicast replicator. In this section, we describe the modifications needed to allow multicast packet transmission in MPLS networks without implementing a new protocol. Three main points are to be considered: information tables of MPLS nodes, multicast packet transmission and, join and prune label distribution and releasing. Our major objectif was implementing the simulation with NS of PIM-SM in MPLS networks without major modifications of the unicast MPLS code in NS assuring compatibility between nodes.

4.1 Information tables of MPLS nodes

As mentioned in Section 2, an MPLS node contains three information tables: LIB, PFT, and ERB. To apply the PIM-MPLS proposition, a mapping of the (S, G) session and the <incoming label, incoming interface> on one hand, and a mapping of the <incoming label, incoming interface> to more than one <outgoing label, outgoing interface>, on the other hand, are needed. The information base at the MPLS nodes must be modified.

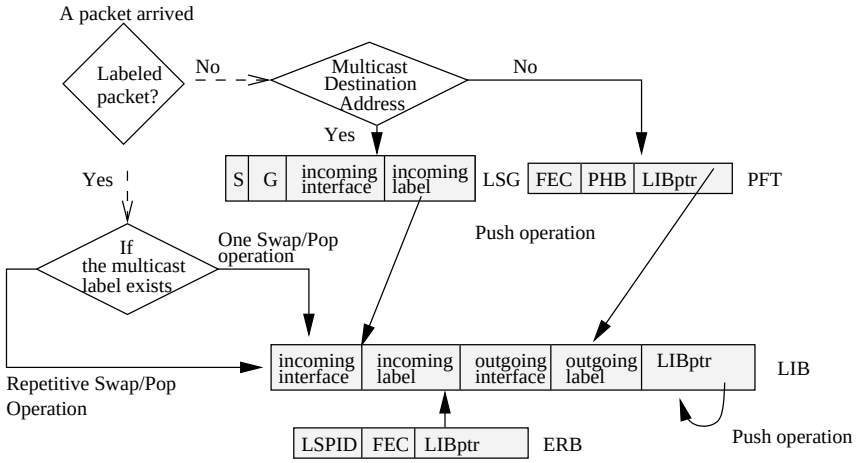


Figure 6: MPLS multicast routing table entries for (S1, G1)

For the first mapping, the Label for Source and Group table (LSG) is defined. This table includes four fields: Incoming label, Incoming interface, Source, and Group. When a new member joins an (S,G) session, and new labels are being allocated upstream, this table is filled at each node. As for the second mapping, where one <incoming label, incoming interface> may be mapped to more than one <outgoing label, outgoing interface> in a branching node, there is no need to create a new table. The LIB table could be filled more than one time with the same <incoming label, incoming interface> but different <outgoing label, outgoing interface>.

In order to insert, remove, upgrade, access, and read the entries in the LSG table, the following functions have been specified:

- int MPLSAddressClassifier::LSGinsert(int iIface, int iLabel, int Source, int Group)

  This function inserts a new entry in the LSG table. The entry must contain the four fields, which are the incoming interface, incoming label, source address, and group address. It is used when a new member joins an (S, G) session.

- int MPLSAddressClassifier::LSGdelete(int Source, int Group)

  This function deletes the LSG entry for an (S, G) session. It is used when a member leaves the (S, G) session.

- int MPLSAddressClassifier::LSGlabellookup(int iIface, int iLabel, int &Source, int &Group)

  This function is used to determine if an (S,G) entry exist for a labeled packet. If an entry exists, then the packet is treated as a multicast packet otherwise it is treated as a unicast packet.

- int MPLSAddressClassifier::LSGSGlookup(int &iIface, int &iLabel, int Source, int Group)

  This function returns the incoming interface and label for an (S,G) session. It is used to check if the node is a tree node (when a join message has been forwarded to the source). If it is the case, than the join message forwarding will stop and the node is considered as a branching one. The incoming interface and label are used to insert another entry in the LIB table but with a new outgoing interface and label.

- void MPLSAddressClassifier::LSGdump(const char *id)

  This function is used for displaying the contents of the LSG table, when required by the user.

4.2 Multicast packet transmission

Data is transmitted exactly as in unicast MPLS packets with only one difference at branching nodes. The procedure is done as follows: When a labeled packet arrives, a search is done in the LSG for the <incoming label, incoming interface>. If the result is positive, then the labeled packet is a multicast packet. Note that this checking can be bypassed but in this case the MPLS unicast code should be changed.

Therefore the node may be a branching one and the LIB table may contain more than outgoing entry. In this case, instead of accessing the LIB table only one time, there must be search in it for more than one entry. For each entry, a packet copy is created, and then label swapped with the corresponding the outgoing label, and then transmitted to the outgoing interface.

Note that:

- There is no real replicator defined at each node. Instead the packet duplication is done in a virtual manner. For each outgoing entry in the LIB table (for the same incoming interface and label), a label swap is done for a copy of that packet, and then this copy is sent on the the outgoing interface.

- These modifications do not affect the unicast MPLS data forwarding, they are executed only when multicast packets arrives.

4.3 Join and prune Label distribution and releasing

The join-group and prune-group functions are two functions executed at nodes that wish to join or to leave an (S,G) session.

The label allocation is done from the joining node toward the source (the join-group function definition is present in the appendix). The join-group algorithm first checks if the node is an

(S, G) session node. If it is, then there is no need to continue the joining process. If not, the algorithm generates new <incoming interface, incoming label> for this node and seeks for the nexthop node toward the source. It installs an entry with the corresponding <incoming interface, incoming label> and associates it with <outgoing interface=-1, outgoing label=-1> to the LIB table since it is the joining node. It installs also an entry to the LSG table with the corresponding <incoming interface, incoming label> and associates it with the (S, G) session. The <incoming interface, incoming label> for this node equals the <outgoing interface, outgoing label> for the nexthop node toward the source. If this nexthop node is an (S, G) session node, then the algoritm searches for the <incoming interface, incoming label> in the LSG table (associated with the (S, G) session) for this nexthop node and inserts an entry to its LIB table with <incoming interface, incoming label> associated to the <outgoing interface, outgoing label> already calculated. This process is repeated at each node toward the source. It should be noted that at the source there is no need for a new incoming interface and incoming label.

When a node prunes itself from a session, some labels must be deallocated. The label deallocation (the prune-group function definition is present in the appendix) is processed at all nodes on the way from the pruned node toward the source. It stops in one of two cases, either when it reaches the source, or when it reaches a branching node for the (S, G) session. Label deallocation means deleting the corresponding (S, G) entries from the LSG and LIB tables. At a branching node, the algorithm deletes the corresponding entry from the LIB table only since the branching node needs the LSG entry to be able to send data packets to other branches.

4.4 Simulation Scenario

Fig. 7 illustrates a simulation example for the PIM-SM protocol (The Tcl file for this example MPLSPIMSMexample.tcl is presented in the appendix). Let's take LSR5 as the source and the

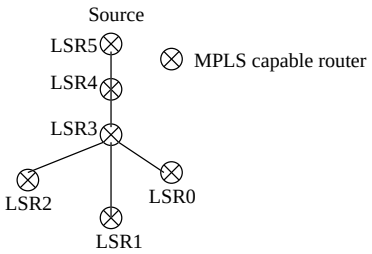


Figure 7: PIM-SM Simulation Example

group address is Group=8 and suppose that LSR0 and LSR1 join the (S, G) session before the source starts its transmission. Let's suppose also that LSR5 is a unicast source at the same time and sends separately unicast packets to LSR0 and LSR2. The MPLS labeling should be automatically done, and all information tables are filled. At T0 the source starts its multicast and unicast data transmissions. While the source is transmitting its packets, LSR0 leaves at T1 the session by executing the prune-group function, and at T2, LSR2 joins the session. The source stops sending multicast packets at T3 and restart transmission at T4. As in unicast, in order to trace the labeling tables, one can use the dump function LSGdump. Note that the LIB table is used for both unicast and multicast transmissions.

4.5 Simulator Evaluation

In order to see how labels are allocated at each node, we will consider the two nodes LSR4 and LSR3, and see how the LIB and LSG tables are filled at each node. At node LSR4, the LSG is filled with the (S = 5, G = 8) entry where <incoming label=6, incoming interface=5>. In the LIB table the <incoming label=6, incoming interface=5> points toward the <outgoing label=6, outgoing interface=3>. This is shown in the next two tables:

```
     ___LIB dump___ [LSR: 4]
-------------------------------------------
     #     iIface     iLabel   oIface    oLabel    LIBptr
     0:      -1         1       3         0         -1
     1:      -1         2       5         0         -1
     2:      -1         3       3         2         -1
     3:      -1         4       3         3         -1
     4:      -1         5       3         4         -1
     5:       5         6       3         6         -1

    ___LSG dump___ [LSR: 4]
-------------------------------------------
     #     iIface    iLabel   Source     Group
     0:      5         6       5         8
```

While for LSR3, which is a branching node, the LSG is filled with the (S=5, G=8) entry where <incoming label=6, incoming interface=4>, and the LIB maps the <incoming label=6, incoming interface=4> entry toward two outputs, one <outgoing label=1, outgoing interface=1> and one <outgoing label=1, outgoing interface=2>. This is shown in the next two tables :

```
     ___LIB dump___ [LSR: 3]
-------------------------------------------
     #     iIface     iLabel   oIface    oLabel    LIBptr
     0:      -1         1       4         0         -1
     1:      -1         2       0         0         -1
     2:      -1         3       1         0         -1
     3:      -1         4       2         0         -1
     4:      -1         5       4         2         -1
     6:       4         6       1         1         -1
     7:       4         6       2         1         -1

    ___LSG dump___ [LSR: 3]
-------------------------------------------
     #     iIface    iLabel   Source     Group
     0:      4         6       5         8
```

Fig. 8 shows the total used bandwidth (unicast packets for LSR0 and LSR1 plus multicast packets for the session (S, G) received from the source LSR5) on link LSR3-LSR4.

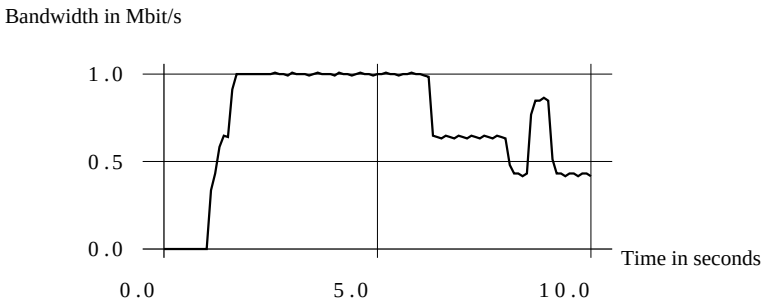


Figure 8: Bandwidth (in Mbit/s) used on link LSR3-LSR4

It is clear from this example that our simulator can be useful for researchers to simulate and evaluate their MPLS multicast and multicasting related techniques.

5 Conclusion

Merging the MPLS technology and the multicast technolgy is very important. Multicasting over MPLS networks can benefit from the multicast reducing of traffic on one hand, and MPLS flexibility, speed and quality of service on the other hand. In this paper, we propose a simulator for multicast routing over an MPLS network by extending MPLS Network Simulator (MNS). Our basic idea is to preserve the existing code for unicast transmission simulation using the M-PLS networks simulator (MNS). Unicast label distribution, LSP construction and L2 switching still functioning the same.

Many multicast protocols have been proposed and are in use today on the Internet. The implementation of multicasting over MPLS networks must be done for each one of these multi-cast protocols. This work focuses on the study of the PIM-SM protocol (source specific mode) over MPLS networks since it is the most widely implemented protocol but our work can be beneficial to other protocols as well. This simulator is based on the piggybacking proposition [13] where a label is piggybacked over the join message in PIM-SM protocol.

The implementation of PIM-SM protocol (source specific tree) over an MPLS network is done with minimum modifications of the unicast MPLS code in NS. A new information table (LSG) which maps the incoming label to an (S,G) session is created. The structure of the multicast packet has the structure of a unicast packet but the MPLS node uses its LSG table to discover from the IP destination address whatever this packet is a multicast packet or not. The LSG entries of the MPLS nodes in an MPLS network are filled and deleted when new group members join or leave a multicast (S,G) session.

The LIB table remains the same when an <incoming label, incoming interface> is mapped to an <outgoing label, outgoing interface>. In branching nodes of multicast trees, an <incoming label, incoming interface> is mapped to more than one <outgoing label, outgoing interface>. The L2 switching is done in the same manner as in unicast MPLS transmission, the only difference is on branching nodes where the MPLS node makes several copies of the packet and

swaps the convenient outgoing label and the packet is transmitted to the nexthop which is associated to the outgoing interface.

There remains a lot more capabilities to be added and extended to the proposed simulator such other MPLS multicast propositions and protocols, multicast trees construction using explicit routes and, QoS support on each node. This simulator can efficiently help researchers to simulate and evaluate their MPLS multicast and multicasting related techniques.

6 Appendix

- MPLS simulation example: file MPLSexample.tcl

```
set ns [new Simulator]

set nf [open MPLSexample.nam w]
$ns namtrace-all $nf
set f0 [open MPLSexample.tr w]
$ns trace-all $f0

proc finish {} {
        global ns nf f0
        $ns flush-trace
        close $nf
        close $f0
        exec nam MPLSexample.nam &
        exit 0
}

$ns rtproto DV

set Node0   [$ns node]

$ns node-config -MPLS ON
set LSR1  [$ns node]
set LSR2  [$ns node]
set LSR3  [$ns node]
set LSR4  [$ns node]
set LSR5  [$ns node]
set LSR6  [$ns node]
$ns node-config -MPLS OFF

set Node7   [$ns node]
set Node8   [$ns node]

$ns duplex-link $Node0 $LSR1  1Mb  10ms DropTail
$ns duplex-link $LSR1 $LSR2   1Mb  10ms DropTail
$ns duplex-link $LSR1 $LSR4   1Mb  10ms DropTail
$ns duplex-link $LSR2 $LSR3   1Mb  10ms DropTail
$ns duplex-link $LSR3 $LSR4   1Mb  10ms DropTail
$ns duplex-link $LSR3 $LSR6   1Mb  10ms DropTail
$ns duplex-link $LSR4 $LSR5   1Mb  10ms DropTail
$ns duplex-link $LSR5 $LSR6   1Mb  10ms DropTail
$ns duplex-link $LSR6 $Node7  1Mb  10ms DropTail
$ns duplex-link $LSR6 $Node8  1Mb  10ms DropTail
```

```
for {set i 1} {$i < 7} {incr i} {
        set a LSR$i
        for {set j [expr $i+1]} {$j < 7} {incr j} {
                set b LSR$j
                eval $ns LDP-peer $$a $$b
        }
        set m [eval $$a get-module "MPLS"]
        $m enable-reroute "new"
}

$ns ldp-request-color       blue
$ns ldp-mapping-color       red
$ns ldp-withdraw-color      magenta
$ns ldp-release-color       orange
$ns ldp-notification-color  yellow

[$LSR1 get-module "MPLS"] enable-control-driven
[$LSR2 get-module "MPLS"] enable-control-driven
[$LSR3 get-module "MPLS"] enable-control-driven
[$LSR4 get-module "MPLS"] enable-control-driven
[$LSR5 get-module "MPLS"] enable-control-driven
[$LSR6 get-module "MPLS"] enable-control-driven

set Src [new Agent/CBR]
$ns attach-agent $Node0 $Src
$Src set packetSize_ 200
$Src set Interval_ 0.08

set Dst [new Agent/Null]
$ns attach-agent $Node7 $Dst

$ns connect $Src $Dst

$ns at 0.5 "$Src start"
$ns at 1.5 "$Src stop"

[$LSR1 get-module "MPLS"] trace-ldp
[$LSR1 get-module "MPLS"] trace-mpls

$ns at 2.0 "[$LSR1 get-module "MPLS"] pft-dump"
$ns at 2.0 "[$LSR1 get-module "MPLS"] lib-dump"
$ns at 2.0 "[$LSR1 get-module "MPLS"] erb-dump"

$ns at 2.0 "[$LSR2 get-module "MPLS"] pft-dump"
$ns at 2.0 "[$LSR2 get-module "MPLS"] lib-dump"
$ns at 2.0 "[$LSR2 get-module "MPLS"] erb-dump"
```

```
$ns at 2.0 "[$LSR3 get-module "MPLS"] pft-dump"
$ns at 2.0 "[$LSR3 get-module "MPLS"] lib-dump"
$ns at 2.0 "[$LSR3 get-module "MPLS"] erb-dump"

$ns at 2.0 "[$LSR4 get-module "MPLS"] pft-dump"
$ns at 2.0 "[$LSR4 get-module "MPLS"] lib-dump"
$ns at 2.0 "[$LSR4 get-module "MPLS"] erb-dump"

$ns at 2.0 "[$LSR5 get-module "MPLS"] pft-dump"
$ns at 2.0 "[$LSR5 get-module "MPLS"] lib-dump"
$ns at 2.0 "[$LSR5 get-module "MPLS"] erb-dump"

$ns at 2.0 "[$LSR6 get-module "MPLS"] pft-dump"
$ns at 2.0 "[$LSR6 get-module "MPLS"] lib-dump"
$ns at 2.0 "[$LSR6 get-module "MPLS"] erb-dump"

$ns at 4.0 "finish"
$ns run
```

- The join message algorithm

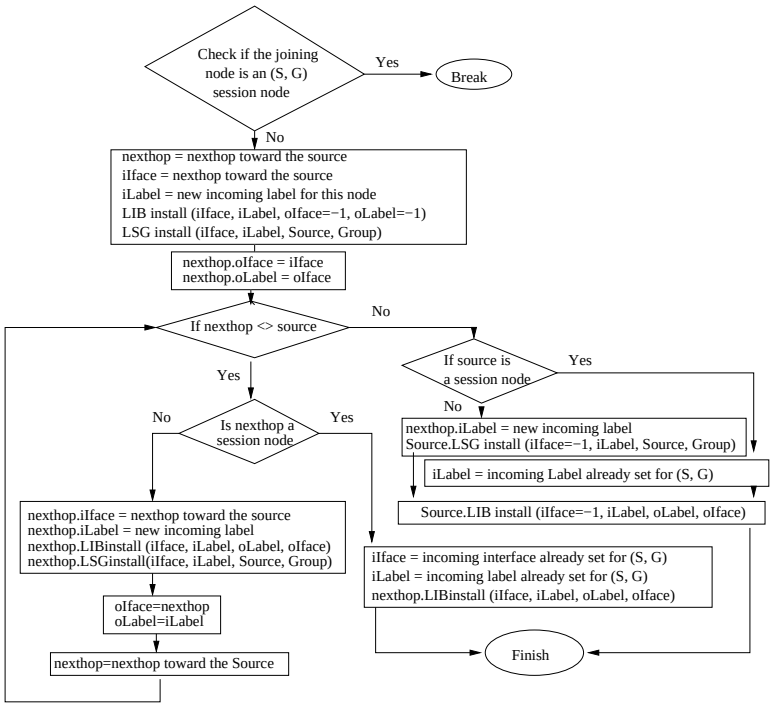


Figure 9: Join-group Algorithm

- The join-group function written in file ns-mpls-node.tcl

```
RtModule/MPLS instproc join-group { Source Group } {
```

```
set Sourceid [$Source id]
set nodeid [[$self node] id]
set node [[Simulator instance] get-node-by-id $nodeid]
set checkSG [$self checkSG $Sourceid $Group]

if {$checkSG != -1} {
  return ``-1''
   }
set oIface $nodeid
set nexthopid [$self lookup-nexthop $nodeid $Sourceid]
set nexthop [[Simulator instance] get-node-by-id $nexthopid]
set iIface $nexthopid
set iLabel [$self new-incoming-label]
set oLabel $iLabel
$self installLIB $iIface $iLabel -1 -1
$self installLSG $iIface $iLabel $Sourceid $Group
for {
set nexthopid [$self lookup-nexthop $nodeid $Sourceid]
set nexthop [[Simulator instance] get-node-by-id $nexthopid]
} { $nexthopid != $Sourceid } {
set nexthopid [$self lookup-nexthop $nexthopid $Sourceid]
set nexthop [[Simulator instance] get-node-by-id $nexthopid]
} {
set iLabel [[$nexthop get-module "MPLS"] get-iLabel-for-SG $Sourceid $Group]
set iIface [[$nexthop get-module "MPLS"] get-iIface-for-SG $Sourceid $Group]
if {$iLabel < 0}
set iLabel [[$nexthop get-module "MPLS"] new-incoming-label]
set iIface [$self lookup-nexthop $nexthopid $Sourcei
[$nexthop get-module "MPLS"] installLIB $iIface $iLabel $oIface $oLabel
[$nexthop get-module "MPLS"] installLSG $iIface $iLabel $Sourceid $Group
} else {
[$nexthop get-module "MPLS"] installLIB $iIface $iLabel $oIface $oLabel
     return
       }
set oLabel $iLabel
set oIface $nexthopid
       }
if {$nexthopid == $Sourceid } {
set check [[$nexthop get-module "MPLS"] checkSG $Sourceid $Group]
if {$check == -1 } {
set iLabel [[$nexthop get-module "MPLS"] new-incoming-label]
[$nexthop get-module "MPLS"] installLSG -1 $iLabel $Sourceid $Group
        } else
set iLabel [[$nexthop get-module "MPLS"] get-iLabel-for-SG $Sourceid $Group]
        }
        [$nexthop get-module "MPLS"] installLIB -1 $iLabel $oIface $oLabel
```

```
        return
        }
}
```

- The prune message algorithm

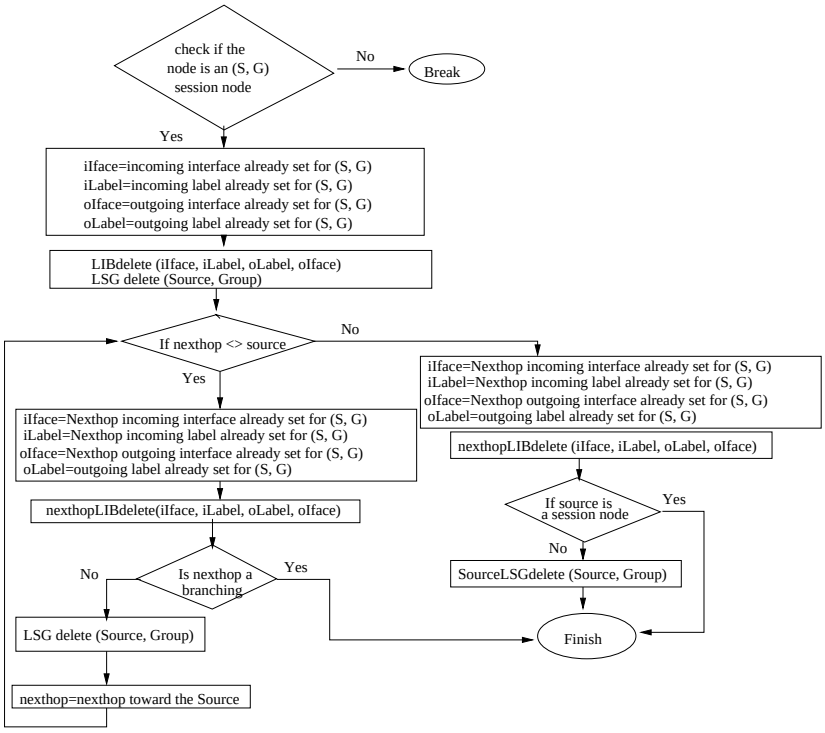


Figure 10: Prune-group Algorithm

- The prune-group function written in file ns-mpls-node.tcl

```
RtModule/MPLS instproc prune-group { Source Group } {

set Sourceid [$Source id]
set nodeid [[$self node] id]
set node [[Simulator instance] get-node-by-id $nodeid]
set nexthopid [$self lookup-nexthop $nodeid $Sourceid]
set nexthop [[Simulator instance] get-node-by-id $nexthopid]
set checkSG [$self checkSG $Sourceid $Group]

if {$checkSG == -1} {
     return "-1"
   }
set iLabel [$self get-iLabel-for-SG $Sourceid $Group]
set iIface [$self get-iIface-for-SG $Sourceid $Group]
set oLabel [$self get-oLabel-for-SG $Sourceid $Group]
```

```
set oIface [$self get-oIface-for-SG $Sourceid $Group]
if {$oLabel != -1} {
      return "-1"
                  }
$self LIBdelete $iLabel $oIface
$self LSGdelete $Sourceid $Group

for {
set nexthopid [$self lookup-nexthop $nodeid $Sourceid]
set nexthop [[Simulator instance] get-node-by-id $nexthopid]
} { $nexthopid != $Sourceid } {
set nexthopid [$self lookup-nexthop $nexthopid $Sourceid]
set nexthop [[Simulator instance] get-node-by-id $nexthopid]
} {
set iIface [[$nexthop get-module "MPLS"] get-iIface-for-SG $Sourceid $Group]
set iLabel [[$nexthop get-module "MPLS"] get-iLabel-for-SG $Sourceid $Group]
set oLabel [[$nexthop get-module "MPLS"] get-oLabel-for-SG $Sourceid $Group]
set oIface [$self lookup-nexthop $nexthopid $nodeid]
[$nexthop get-module "MPLS"] LIBdelete $iLabel $oIface
set checkbranching [[$nexthop get-module "MPLS"] is-branching $Sourceid $Group]
        if { $checkbranching > 0} {
                return "1"
        }
        $self LSGdelete $Sourceid $Group
        }

set iIface [[$nexthop get-module "MPLS"] get-iIface-for-SG $Sourceid $Group]
set iLabel [[$nexthop get-module "MPLS"] get-iLabel-for-SG $Sourceid $Group]
set oLabel [[$nexthop get-module "MPLS"] get-oLabel-for-SG $Sourceid $Group]
set oIface [$self lookup-nexthop $nexthopid $nodeid]

        [$nexthop get-module "MPLS"] LIBdelete $iLabel $oIface

set checksourcebranching [[$nexthop get-module "MPLS"]
is-branching $Sourceid $Group]
        if { $checksourcebranching > 0} {
                 return "1"
        }
[$nexthop get-module "MPLS"] LSGdelete $Sourceid $Group
}
```

- MPLS PIM-SM simulation example: file MPLSPIMSMexample.tcl

```
set ns [new Simulator]
set nf [open MM10.nam w]
$ns namtrace-all $nf
set f0 [open MM10.tr w]
```

```
$ns trace-all $f0

proc finish {} {
        global ns nf f0
        $ns flush-trace
        close $nf
        close $f0
        exec  nam MM10.nam &
        exit 0
}

$ns rtproto DV

$ns node-config -MPLS ON
set LSR0  [$ns node]
set LSR1   [$ns node]
set LSR2  [$ns node]
set LSR3  [$ns node]
set LSR4  [$ns node]
set LSR5  [$ns node]
$ns node-config -MPLS OFF

$ns duplex-link $LSR0 $LSR3  1Mb  10ms DropTail
$ns duplex-link $LSR1 $LSR3  1Mb  10ms DropTail
$ns duplex-link $LSR2 $LSR3  1Mb  10ms DropTail
$ns duplex-link $LSR3 $LSR4  1Mb  10ms DropTail
$ns duplex-link $LSR4 $LSR5  1Mb  10ms DropTail


for {set i 0} {$i < 6} {incr i} {
        set a LSR$i
        for {set j [expr $i+1]} {$j < 6} {incr j} {
                set b LSR$j
                eval $ns LDP-peer $$a $$b
        }
        set m [eval $$a get-module "MPLS"]
        $m enable-reroute "new"
}

$ns ldp-request-color       blue
$ns ldp-mapping-color       red
$ns ldp-withdraw-color      magenta
$ns ldp-release-color       orange
$ns ldp-notification-color  yellow

[$LSR0 get-module "MPLS"] enable-control-driven
[$LSR1 get-module "MPLS"] enable-control-driven
```

```
[$LSR2 get-module "MPLS"] enable-control-driven
[$LSR3 get-module "MPLS"] enable-control-driven
[$LSR4 get-module "MPLS"] enable-control-driven
[$LSR5 get-module "MPLS"] enable-control-driven


set Group 8

set Src [new Agent/CBR]
$ns attach-agent $LSR5 $Src
$Src set packetSize_ 200
$Src set Interval_ 0.08
$Src set dst_addr_ $Group
$Src set dst_port_ 0

$ns at 0.5 "[$LSR0 get-module "MPLS"] join-group $LSR5 $Group"
$ns at 0.5 "[$LSR1 get-module "MPLS"] join-group $LSR5 $Group"

$ns at 1.0 "$Src start"
$ns at 1.2 "[$LSR0 get-module "MPLS"] prune-group $LSR5 $Group"
$ns at 1.3 "[$LSR2 get-module "MPLS"] join-group $LSR5 $Group"
$ns at 1.5 "$Src stop"

$ns at 2.5 "[$LSR0 get-module "MPLS"] pft-dump"
$ns at 2.5 "[$LSR0 get-module "MPLS"] lib-dump"
$ns at 2.5 "[$LSR0 get-module "MPLS"] lsg-dump"

$ns at 2.5 "[$LSR1 get-module "MPLS"] pft-dump"
$ns at 2.5 "[$LSR1 get-module "MPLS"] lib-dump"
$ns at 2.5 "[$LSR1 get-module "MPLS"] lsg-dump"

$ns at 2.5 "[$LSR2 get-module "MPLS"] pft-dump"
$ns at 2.5 "[$LSR2 get-module "MPLS"] lib-dump"
$ns at 2.5 "[$LSR2 get-module "MPLS"] lsg-dump"

$ns at 2.5 "[$LSR3 get-module "MPLS"] pft-dump"
$ns at 2.5 "[$LSR3 get-module "MPLS"] lib-dump"
$ns at 2.5 "[$LSR3 get-module "MPLS"] lsg-dump"

$ns at 2.5 "[$LSR4 get-module "MPLS"] pft-dump"
$ns at 2.5 "[$LSR4 get-module "MPLS"] lib-dump"
$ns at 2.5 "[$LSR4 get-module "MPLS"] lsg-dump"

$ns at 2.5 "[$LSR5 get-module "MPLS"] pft-dump"
$ns at 2.5 "[$LSR5 get-module "MPLS"] lib-dump"
$ns at 2.5 "[$LSR5 get-module "MPLS"] lsg-dump"
```

```
$ns at 3.0 "finish"
$ns run
```

References

[1] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. IETF RFC3031, January 2001.

[2] G. ahn and W. Chun. Overview of MPLS network simulator: Design and implementation. Chungnam National University, Korea,http://flower.ce.cnu.ac.kr/ fog1/mns.

[3] K. Fall. and K. Varadhan. The NS Manual. UC Berkeley, LBL, USC/ISI, and Xerox PARC, January 2001.

[4] UCB/LBNL/VINT. Network animator. URL: http://www.isi.edu/nsnam/nam.

[5] D. Estrin et al. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. IETF RFC 2362, 1998.

[6] S. Deering et al. Protocol Independent Multicast version 2-Dense Mode Specification. IETF Internet Draft, http://catarina.usc.edu/pim/pimdm/pim-dm-06.txt, 1997.

[7] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas. LDP specification. IETF RFC3036, Januray 2001.

[8] M. Ramalho. Intra- and Inter-domain multicast routing protocols: A survey and taxonomy. *IEEE Communications Surveys and Tutorials*, 3(1):2–25, First Quarter 2000.

[9] H. Holbrook and B. Cain. Source-Specific Multicast for IP (SSM). IETF Internet draft, 2001.

[10] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP tunnels. IETF RFC3209, December 2001.

[11] X. Xiao, A. Hannan, B. Bailey, and L. Ni. Traffic engineering with MPLS in the Internet. *IEEE Network*, 14(2):28–33, March/April 2000.

[12] D. Ooms, W. Livens, A. Acharya, F. Griffoul, and F. Ansari. Framework for IP multicast in MPLS. IETF Internet draft, January 2002.

[13] D. Farinacci, Y. Rekhter, and E. Rosen. Using PIM to distribute MPLS labels for multicast routes. IETF Internet draft, November 2000.

[14] A. Fei, J. Cui, M. Gerla, and M. Faloutsos. Aggregated multicast: An approach to reduce multicast state. In *Proceedings of the Third International COST264 Workshop (NGC 2001) UCL. London*, number 2233 in LNCS, pages 172–188, november 2001.

[15] A. Boudani and B. Cousin. A new approach to construct multicast trees in mpls networks. In *Seventh IEEE Symposium on Computers and Communications*, pages 913–919, 2002.

[16] G. Ahn and W. Chun. Design and implementation of mpls network simulator supporting ldp and cr-ldp. In *IEEE International Conference on Networks (ICON'00)*, 2000.